Aalto University
School of Science
Master's Programme in ICT Innovation

Hugo Allain

Improving productivity and reducing costs of mobile app development with Flutter and Backend-as-a-Service

Master's Thesis
Espoo, October 29, 2020

Supervisors: Professor Mario Di Francesco, Aalto University
Advisor: Professor Mario Di Francesco

Aalto University
School of Science
Master's Programme in ICT Innovation

ABSTRACT OF
MASTER'S THESIS

Author: Hugo Allain

Title:
Improving productivity and reducing costs of mobile app development with Flutter and Backend-as-a-Service

Date: October 29, 2020 **Pages:** 94

Major: Cloud Computing and Services **Code:** SCI3081

Supervisors: Professor Mario Di Francesco

Advisor: Professor Mario Di Francesco

Many solutions exist nowadays for developing mobile applications. Hybrid development is one such solution and allows developers to build a mobile application for multiple platforms using a single code base. Flutter is a framework developed by Google allowing developers to develop hybrid applications. This framework aims to deliver the same performance as native applications. Flutter is a direct competitor to React Native, which is the dominant framework for hybrid development. Alternative tools are available for developers to speed up and make application development easier. Backend-as-a-Service (BaaS) solutions are part of these tools and allow developers to integrate the features most used in mobile development with great ease. These BaaS solutions have several advantages, such as saving time and cost in developing a mobile application. Nowadays, cloud solutions are ubiquitous, and many large companies like Facebook, LinkedIn or Twitch trust BaaS solutions and use these solutions in their architecture.

This thesis presents Alacrity, a hybrid mobile application developed by Red Nuclear Monkey and which uses the Flutter framework. This work shows the solution found by Red Nuclear Monkey to allow the implementation of several BaaS solutions within a Flutter project. Besides, this thesis presents a literary review of academic studies and industry literature to determine the motivations of companies in choosing to use a Backend-as-a-Service solution or a custom backend. Finally, this research suggests the best current BaaS solution for developing a Flutter application.

Keywords: Hybrid applications, Backend-as-a-Service, BaaS, Flutter, Firebase, Parse, iOS, Android, Cloud, Back4App

Language: English

Acknowledgements

I am grateful for the efficient support and guidance of my supervisor and advisor Mario Di Francesco throughout this thesis and. He allowed me to graduate on time despite the complicated conditions imposed by the COVID-19. He contributed to the subject of this thesis thanks to the course he teaches and the assignments he offers, I realized that I was highly interested in mobile development.

I would also like to thank Egor Danchenkov, who allowed me to do my thesis at Red Nuclear Monkey and to complete a thesis on a subject that I wanted by giving me support whenever I needed it. He allowed me to complete my thesis in excellent conditions without ever putting any pressure on me and always offered any kind of help. Besides, he gave me a lot of tips that will be useful to start my professional life.

I wish to thank EIT Digital and Guillaume Pierre, who have allowed me to discover new academic and social experiences through a fantastic and complete program. Thanks to them, I have discovered lots of new people from all over the world with whom I have shared lots of good times, and who have brought me a lot.

Then, I would also like to thank Emilie Lejeune, who indirectly helped me a lot during this work. She supported me daily and gave me a lot of strength to complete this last stage of my studies.

Finally, I would like to thank my parents who have always been there for me and who have followed me in all the choices I have been able to make so far.

Thank you all!

Espoo, October 29, 2020

Hugo Allain

Abbreviations and Acronyms

BaaS	Backend-as-a-Service
MBaaS	Mobile-Backend-as-a-Service
IaaS	Infrastructure-as-a-service
PaaS	Platform-as-a-Service
SaaS	Software-as-a-Service
APIs	Application programming interfaces
OS	Operating System
SDKs	Software development kits
SMS	Short Message Service
REST	Representational state transfer
UI	User Interface
MVC	Model-View-Controller
UML	Unified Modeling Language
OO	Object-oriented
AOT	Ahead Of Time
JIT	Just In Time
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
OEM	Original Equipment Manufacturer
RAM	Random-access memory
MVPs	Minimum Viable Product
OAuth	Open Authentication
IdP	Identity provider

Contents

Chapter 1

Introduction

In the early 2000s, the primary purpose of mobile phones was to communicate by calling or texting an interlocutor. However, nowadays the use of mobile phones is different: with the launch of the iPhone in 2007 and the Apple App Store in 2008[1], mobile phones have seen the rise of new functionalities and seem to have become vital for a large part of its users [32]. Since then, mobile phones have become tools that have changed our world [30], allowing users to entertain themselves, learn, and search for information faster and more efficiently. Most of these features have become obtainable thanks to applications, either directly included when purchasing the mobile phone, or downloadable from application stores such as the Apple App Store or Google Play Store. These stores are both today the largest ones for mobile applications, excluding China where Google Play Store is not officially available [4].

A part of the world population uses several social network applications such as Facebook, Twitter or Instagram in a daily basis. In the first quarter of 2020, 2.6 million users were active per month on Facebook [20], and by April 2020, over 98% of active Facebook users had used it on mobile devices [19]. In 2019, almost half of the current world population, more precisely 3.2 billion people, used a smartphone [34]. With this considerable number of mobile device users growing over the years, the application market has exploded. Indeed, the number of applications available in the various stores is consequently increasing [14]. Besides, according to a study by Red Hat, 50% of today's businesses need a mobile app to serve their interests, which are to make their operations more productive and efficient [33] [35].

Technological advances, particularly in the cloud, have enabled the emer-

[1]The Age of Apps: Evolution of the Mobile Application [Infographic] https://www.industryweek.com/technology-and-iiot/information-technology/article/21958019/the-age-of-apps-evolution-of-the-mobile-application-infographic

gence of solutions, for instance Backend-as-a-Service (BaaS), to facilitate the development of applications. BaaS is a solution that facilitates the construction of web and mobile applications by allowing developers to link their product to a cloud-stored backend via application programming interfaces (APIs) and software development kits (SDKs) [6]. Several businesses, such as Google or Amazon, offer these BaaS solutions with their product named respectively, Firebase and AWS Amplify. BaaS contains a wide range of features that lead to reduced cost and development time. Even if BaaS facilitate the development of mobile applications, certain amount of companies faced a dilemma [35]: should the company develop a native or a hybrid app? A native application is an application built specifically for a single operating system (OS) while a hybrid application uses a single code to run on several platforms. Hybrid applications are debated compared to the native ones, particularly because of their lower performance. Nonetheless, the future of development for mobile applications is turning to hybrid development [35].

1.1 Research topic

Red Nuclear Monkey is a young consulting web and mobile development company, which aims to create, change or renew applications according to customer needs. This company aims to use the latest technologies available on the market and is steadily looking for new products and innovation. Following the rise of an increasingly popular framework called Flutter, the company Red Nuclear Monkey already had the objective of creating a hybrid application using this framework for customers or for the company. With the condition difficulties and restrictions put in place by most European governments regarding COVID-19 in 2020, Red Nuclear Monkey has experienced a break in its activity regarding projects involving clients. Therefore, the company decided to focus on internal projects during this complicated period for a consequent amount of small businesses, and thus renewed the idea of developing a hybrid application using Flutter.

This application is a project-specific to the company, which means there is no involvement regarding clients. The company wants to explore Flutter further throughout the development of the app and produce a complete product available on both the Google Play Store and the Apple App Store. This project plans to be a part of the company's portfolio. Indeed, for a young consulting company, it is meaningful to present the projects carried out to attract more potential customers but also to show the skills of the company and what it can achieve.

As the company has limited means and resources, this project aims to

minimize the cost of developing the application and put this application into production as quickly as possible for the two most common types of operating systems known today, iOS and Android. This app, totally developed by Red Nuclear Monkey, must be regulated so the company can publish it on both stores.

The company chose the Flutter framework because it seems to offer solutions to satisfy the objectives. Therefore, using this framework is not chosen aimlessly since it is nowadays much discussed and seems to have taken precedence over React Native. React Native was the main framework used in the hybrid development of mobile application. To speed up the development process and reduce the cost of the application, the company will couple the functionality of Flutter using several features provided by a Backend-as-a-Service. These features will also fulfill the objectives. However, since Flutter and BaaS involves many new system designs, the company Red Nuclear Monkey needs to make decisions to know which Baas would be the most suitable in building a specific Flutter application.

1.2 Research questions and methodology

Based on the objectives of the company Red Nuclear Monkey explained in the Section 1.1, this paper answers the following questions:

- Which Backend-as-a-Service is the most optimal choice for a company with limited resources when developing a Flutter application?
- What are the most important technical and business indicators while choosing BaaS in general and for this specific use case?

To begin with, this research presents the implementation of the different screens of the application using the Flutter framework. Then, the thesis offers the integration and implementation of the various functionalities, required for the app, of the Backend-as-a-Service used. Finally, this thesis gives a comparison of these BaaS to determine which one of them is potentially the most relevant for Red Nuclear Monkey respecting their restrictions and their objectives. The comparison takes into account the various features that BaaS offer, their implementations and their impact over the company's objectives.

To answer the research questions previously given, this thesis proposes a review of academic literature and industry literature and the implementation of two versions of the Alacrity application.

The literature review will help understand why companies are turning to Backend-as-a-Service solutions rather than developing a custom backend for each application. To carry out this research, this thesis uses academic journals available publicly on electronic bookstores offering professional work such as IEEE, Google Scholar, Aalto doc or Theseus. Besides, this thesis is based on some blog posts that come from platforms offering BaaS solutions.

The two versions of Alacrity each have a different BaaS solution. This work shows, using implementations, which BaaS solution is most effective for the functionality required for Alacrity development. Also, this paper offers a review of the BaaS solutions used thanks to their documentation and official websites. Thus, thanks to the implementation and the data collected through their respective sites, this thesis gives the company Red Nuclear Monkey the ideal solution to use for the development of Alacrity.

1.3 Structure of the thesis

The organization of the thesis is as follows. Chapter 2 presents Flutter, the new dominant framework for hybrid mobile application development. This chapter also presents comparisons: Flutter vs. React Native and Flutter vs. native development. Chapter 3 describes the application developed by Red Nuclear Monkey and the implementation of primary screens. Chapter 4 provides general information about Backend-as-a-Service and the retained BaaS regarding the company's objectives. Chapter 5 offers the implementation of the features of the different BaaS chosen. Chapter 6 gives a comparison between the implementation of the features of the Backend-as-a-Service solutions, discusses the findings followed of comparison and give the most suitable BaaS for this specific case. Finally, Chapter 7 offers concluding remarks for the thesis and directions for future work.

Chapter 2

Hybrid mobile application

As stated in Chapter 1, there are several approaches to developing a mobile application. Three major categories of applications stand out to achieve this purpose: Web, native and hybrid applications. Web and hybrid applications are among the approaches applied for cross-platform development. Cross-platform development is the practice of developing software accessible on several platforms [25]. To develop cross-platform mobile applications and to reduce development cost and time, developers tend to turn to hybrid development [11]. Hybrid apps are a combination of web and natives apps. One of the most significant advantages of hybrid apps is to have a single base code for an app that will then be accessible on Android and iOS. This is one of the principal reasons Red Nuclear Monkey company used hybrid development for its Alacrity application. Having only one code to maintain two mobile operating systems, i.e. iOS and Android, is a considerable advantage given the company's goals. Besides, hybrid applications are easy to build since they use web technologies such as HyperText Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript. To display the HTML and process the JavaScript code of the application, the hybrid approach uses native containers, e.g. WebView for Android and UwebView for iOS [5][11]. However, despite the various advantages offered by the hybrid approach, several disadvantages are also present, such as the performance that is less efficient compared to native applications [11] and the user experience due to the lack of native UI components [5][28].

The next section presents Flutter, a framework for building hybrid mobile applications. This chapter compares Flutter to the most popular framework for hybrid development today, React Native. Also, this chapter presents several types of research to discuss the advantages and disadvantages of Flutter compared to native development.

2.1 Flutter

Flutter is a modern and agile Software Development Kit (SDK) and mobile User Interface (UI) framework for building hybrid apps released in 2018. Flutter is developed by Google and is an open-source SDK, which means the community can help develop and improve the framework. Flutter, like other hybrid app building frameworks, uses a single code base for different platforms, such as web, mobile and Windows. The thesis only covers the mobile development of a Flutter application. Google's goal is to provide a framework capable of improving development speed and offering hybrid applications with similar performance and UI to native applications [25]. Also, Flutter has a different rendering and coding language from other frameworks [28]. In Flutter's case, the programming language used is Dart.

2.1.1 Dart as a programming language

Dart is a programming language released in 2011, developed and maintained by Google. Google aimed to replace the JavaScript programming language with its new Dart language. One of the most significant features of Dart is that the language is both 'Ahead Of Time' (AOT) and 'Just In Time' (JIT). Consequently, Dart avoids JavaScript bridges so that the execution is faster, and proposes a 'Hot reload' feature which allows to speed up the development cycle [39][28]. By using Dart, Flutter does not use Original Equipment Manufacturer (OEM)widgets, i.e. platform system widgets, and thus avoids the performance issues that the other frameworks face by frequently accessing resources from the iOS and Android SDKs platforms [28].

2.1.2 Widgets

Therefore, Flutter provides its own widgets. Flutter widgets refer to all the components required to build the UI. For example, to display a text or a list, the developer can use the Text Widget and ListView. Figure 2.1 compares the HTML/CSS structure to Flutter structure that uses widgets. Unlike other frameworks, Flutter includes in the app the renderer of widgets, using Skia[1], and does not use the system [28]. This Google's framework provides two libraries for the design of widgets: 'Cupertino' for iOS components and 'Material Design' for Android components. Thus, using Flutter, it is possible to have an application with a different UI depending on the platform a customer chooses to use.

[1]Skia in Flutter `https://skia.org/dev/flutter`

Flutter classifies widgets into two categories: Stateless widget and Stateful widget. State refers to data that changes over the life-cycle application. When a widget contains data that can change over time, it will need a rebuild to represent the latest changes on the user's device. Therefore, this widget maintains a state and belongs to the Stateful widget category. Stateless widgets do not have a state and do not require a rebuild.

Figure 2.1: HTML/CSS structure VS Flutter structure.

2.1.3 State management

Stateful widgets can hold data on their own when no other widget needs to use that same data. In Flutter, the documentation calls this approach 'local state'. However, when several parts of an application need the same data and can change it, developers will have to adopt a state management approach to allow the sharing of this data through the different widgets. The idea to put data into a global object so that any widgets can have access to this data. This process is called the 'global state'. Chapter 3 provides more details on State Management and presents the state management approach used by Red Nuclear Monkey for the development of Alacrity and the other techniques available in Flutter.

2.2 Flutter, the new leader of hybrid development?

Even if Flutter is a recent framework, it is becoming one of the most popular frameworks for hybrid application development. As Google trends[2] show, Flutter is growing in popularity compared to its direct competitor, React Native. This implies that people are becoming more and more curious about Flutter. Even if the developers use React Native more than Flutter, they prefer Flutter, according to a survey performed by Stack Overflow in 2019[3], one of the most prominent questions and answers websites for programmers.

The next section presents a comparison of the two frameworks using data collected in several studies. These papers compare the two frameworks to determine which of them is potentially the best by considering several aspects such as performance and user interface.

2.3 Flutter vs React Native

As explained already in this chapter, React Native is still the most widely used framework for cross-platform development of mobile applications in 2020[4]. Facebook created React Native and officially launched this framework in 2015. This framework is also open-source. In terms of programming language, React Native uses JavaScript. Several popular applications use React Native, such as Facebook, Instagram or Uber Eats. Since the launch of Flutter, React Native faces a debate to know which of these two frameworks is further attractive and complete in building hybrid apps to compete with native apps.

First, React Native is older compared to Flutter, and it has a larger community. Thus, there are several solutions for different programming problems, i.e. the React Native community provides more libraries than Flutter's [5][42]. Besides, by using JavaScript as a programming language, React Native is accessible by a larger number of developers, compared to Flutter,

[2]Comparison between 'React Native' and 'Flutter' terms for the past 12 months `https://trends.google.com/trends/explore?cat=5&q=React%20Native,Flutter`

[3]Developers survey results 2019 - Stack Overflow `https://insights.stackoverflow.com/survey/2019#overview`

[4]Cross-platform mobile frameworks used by software developers worldwide in 2019 and 2020 - Statista `https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/`

which uses Dart. This language factor can be decisive in choosing a framework for a company, particularly if a company does not have the time and the means to invest in training employees in a new programming language. It is more interesting to reuse the skills of developers to reduce the cost and speed up the development [5]. However, since React Native uses JavaScript, it faces the performance problem pointed out earlier in this chapter, since it frequently accesses resources from Android and iOS platforms, especially for rendering OEM widgets using JavaScript bridges [5]. Therefore, Flutter has a better performance for rendering widgets [42]. Regarding the user interface, Flutter stands out with comprehensive UI widgets compared to the minimal widgets proposed by React Native. Flutter allows developing a specific graphical interface for each platform more efficiently and quickly than React Native, which requires the use of third-party libraries [5][38]. By providing its own widgets, Flutter takes advantage in code reuse. Reusing code in Flutter appears to be faster, more efficient and more flexible [38].

According to these studies, both frameworks have their strengths. However, it seems that in the future Flutter will become the most used framework considering its evolution and its success during the last few months [5][38].

After comparing Flutter to its direct competitor, React Native, it is worth knowing where Flutter applications stand compared to native applications. As a reminder, Google created Flutter to offer a hybrid development framework offering similar or close performance to native applications. The next section presents research conducted to compare the distinct features of Flutter's apps to Android and iOS apps.

2.4 Flutter vs Native development

When it comes to developing simple apps, the study conducted by M.Olsson [25] reports that the Flutter app performs similarly to Android and iOS apps. The performance appears to be identical when the application is basic, involving few animations and a basic user interface. Since the application uses more complex and heavy animations, the performance of Flutter apps is much lower than the native applications' [16]. Performance includes several parameters, such as interaction with the phone API, rendering speed and business logic [15]. This explains why, in some studies, Flutter applications are perceived to be as efficient as native applications. Regarding code, Flutter requires fewer lines of code and applies the reuse code principle more often compared to native application development [25][26]. Flutter does not separate functional code from UI code, which significantly reduces

the number of lines and files concerned in the application project [25]. However, even though Flutter applications require a reduced amount of code, the size of the final build are significantly larger than the size of the final build of native applications, and the installed applications seem to require more Random-access memory (RAM) compared to native applications [26]. Regarding the UI, studies conducted by M.Olsson [25] and O.Dahl [28] show that users distinguish the Flutter app from native apps. Users seem to make this distinction with the animations of the apps. Flutter applications do not provide by default identical animations to native's animations. Thus, developers require extra work on animations to get an application that has the same behaviour as a native application. Moreover, the study directed by O.Dahl [28] shows that users make also this distinction thanks to the responsiveness of the native application, e.g. Android in this case, which they perceive to be faster than the Flutter application.

To conclude, for the realization of simple applications, Flutter can be a powerful tool for developing applications similar to native applications. For example, for producing Minimum Viable Products (MVPs), companies can see Flutter as the ideal tool, as it allows to quickly and efficiently develop applications accessible to both iOS and Android platforms. All studies presented around Flutter are optimistic for its future and believe that this framework may become the next primary tool for hybrid and native application development. However, when the application becomes more complex, Flutter seems to have several negative behaviours, particularly related to performance and its animations, which may differ from native animations.

Chapter 3

Alacrity: a Flutter application example

This section introduces Alacrity, the mobile application described in Chapter 1. Alacrity is a hybrid mobile and internal application developed by Red Nuclear Monkey. Therefore, the development of the application does not involve any external customer. This project concerns two employees who are also responsible to determine the final expectations of Alacrity. The application is using Flutter as a framework and is the first concrete mobile application for the company. As Flutter is new for Red Nuclear Monkey, the company decided not to create a compound application. Thus, this application does not require complex factors such as creative graphics, heavy computations and network features.

Alacrity is a time management and productivity mobile application. The app is for people who struggle to manage their time, achieve their goals, individuals who procrastinate, and who want to optimize their time to increase their productivity as much as possible. Alacrity makes it easier for users to reach their aim, while also rewarding them. Each goal contains the person's motivations and distractions for completing that task. As a user gets closer to their goal, they will earn a specific amount of points. The points allow users to earn rewards and get additional in-app content by spending them in the shop feature of the application and are winnable when users complete sessions. A session is a period spent on the goal and chosen by the user. Users can achieve a goal with one or multiple sessions, depending on the goal they set. For example, if a user has two goals: Learning Finnish and Respond to an email, the first goal will require more than one session compared to the second goal needing only one. Therefore, users are free to choose the time of their sessions, i.e. the time they want to spend on their goal at the moment. Besides, the users can decide if they prefer to start a

'full' session that lasts 2 hours and which requires at least 1 hour to complete or start a 'bootstrap' session that also lasts 2 hours but only requires at least 15 minutes to complete. Bootstrap sessions give a fewer amount of points. When a session is over, users can check off the distractions of their goals if they succeed to avoid these distractions. Avoiding distractions allows users to earn more points. Further, a user can cancel a session before finishing it, but he will gain a reduced amount of points.

Example: The goal of a person is to learn the basics of Finnish by September 30, 2020. This user has two motivations, 'Learning a new language' and 'a better integration in Finland'. Social media, music and videos are distractions for this individual. Now, this user starts a two hours session to devote this time to his goal. After the two hours have passed, the user indicates that social media was the only distraction he did not avoid. Then he earns a certain number of points depending on the length of his sessions and the two distractions he avoided, i.e. music and videos.

3.1 Main screens for research

This thesis does not cover the entire development of the application. It only presents the necessary parts for the use of the functionalities of Backend-as-a-Service and as several parts of the application use the same BaaS functionalities, such as writing and reading databases, this research focuses on three screens:

- the **Authentication** screen, which illustrates the registration and the login of a user,
- the **Goals** screen, that contains the user's goals list and where it is possible to create and edit a new goal,
- and the **Profile** screen, which displays the data of the current user and the log out.

This chapter presents the widget composition of the UI of these three screens. Implementation of functions using the features of Backend-as-a-Service are present in Chapter 5. The code developed to build the application's UI is the same for each new BaaS tested since it does not contain any development related to the functionality provided by the BaaS. This allows the development of the UI without worrying about the backend implementation.

3.2 Structure of the application

Alacrity is the first Flutter application developed by Red Nuclear Monkey. Therefore, the development team had to agree on the structure of the application before starting to implement it. This section presents the structure of the application, including the design pattern used by the company for the system design and the flow of the app between the three screens presented previously, i.e. Authentication, Goals and Profile.

3.2.1 Design pattern: Model-View-Controller (MVC)

In software development, a design pattern is a general and reusable solution for the repeated and most common problems faced during system design [27]. Pattern designs do not include code. They are templates or descriptions illustrating a solution to a specific problem [27]. Developers enhance these design patterns and make them evolve. Over time, the patterns become more robust and flexible solutions [12].

Regarding the structure of the Alacrity application, the company used a closed approach to the Model-View-Controller (MVC) pattern, which is one of the most eminent and used patterns, in particular in the software architecture of interactive systems [23]. This pattern separates an application into different layers each having a specific role. The MVC pattern defines the role of each of these sections and the means of communication between them [7][18]. The MVC pattern has three components:

1. The first layer is the **Model**. This component receives and manipulates data. It interacts with the database, provided here by BaaS solutions, and transmits this data to the Controller which is another component of the MVC model.

2. The second layer is the **View**. The View component contains the information that the user will get, which is the user interface. For example, in web applications, the View usually comprises HTML and CSS. This component also communicates with the Controller, because it allows exposing the data retrieved by the Controller, and makes it visible and accessible to the user.

3. Finally, the last layer is the **Controller**. The Controller receives input from the user, for example, when a user clicks on a link and makes a request. The Controller, therefore, processes the requests made by

a user. It communicates with both the Model to retrieve the data necessary to satisfy the request and the View to give the data and thus allows the user to receive the response to his request.

The Red Nuclear Monkey development team agreed to rename two components of the MVC model to be more consistent in the environment of a mobile application and Flutter. Manager refers to the Controller component, and Screen refers to the View component. Thus, the company split the Controller layout between View and Managers, so that this approach differs from the MVC pattern. The idea was to keep the separation between models and business logic. Models handle the database logic, and Managers handle the business logic and a live query with the database. Figure 3.1 illustrates a part of the organization of files within the Alacrity project. As this chapter only illustrates the general structure of the application, this section does not discuss the attributes and implementation details of models and controllers. However, Chapter 5 gives a detailed implementation and explanation of these components. Regarding the organization of the files, the Manager file contains the controllers for the sessions and the top-goals of a user. Top-goals are the person's objectives. In this project, the managers are also in association with the BaaS database, because they make it possible to retrieve in real-time data related to sessions and top-goals. Thus, managers allow the view to manage dynamically new data and make it visible to users. Besides, thanks to this interaction, the Screen and Model components do not communicate with each other. The Models file contains all the models used by Alacrity:

- a User class model, which represents a user with all his information and attributes, such as the user identifier (UID),
- a Session class, allowing to represent the sessions of a user, i.e. each period when the user fulfils his top-goal,
- and a TopGoal class, which is a representation of a user's goal.

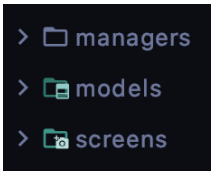


Figure 3.1: Files organization for the close approach of MVC model.

Finally, the Screen file contains all the views of the application. The UI development is only present in these files.

Figure 3.2 illustrates how the close approach of the MVC design model handles a request. The request is a concrete use case of the Alacrity app. Figure 2 describes the following use case:

1. The user creates a new top-goal interacting with the Screen (View). It sends a request to the application to create this new top-goal.
2. The application will go through the Top-goals Manager for this request.
3. The manager will then ask the model to create the top-goal with the information entered by the user.
4. The model will perform a regular query to create a new top-goal with the information entered by the user.
5. Two cases are possible:
   - The answer is successful: the Manager receives the new top-goal because it maintains a live query with the BaaS database, and adds the new-top-goal to the user's list of goals.
   - The answer is unsuccessful: the Manager does not get any new information from the live query so it keeps the 'old' list of goals.
6. The Manager sends its current list of goals to the Screen Goal, either a new one containing the last goal created by the user, or the old one if an error occurred.
7. The Screen renders the top-goals list provided by the Manager, e.g. Controller, using Flutter widgets. The screen of the user contains either:
   - all his top-goals including the last top-goal he created,
   - or all his previous top-goals and an error message stating that there was an error creating his last top-goal.

Once the company agreed on the design pattern choice and the organization of the files for the Alacrity application development project, it was necessary to establish the behaviour of the application and the interaction that the different parts would have with each other.

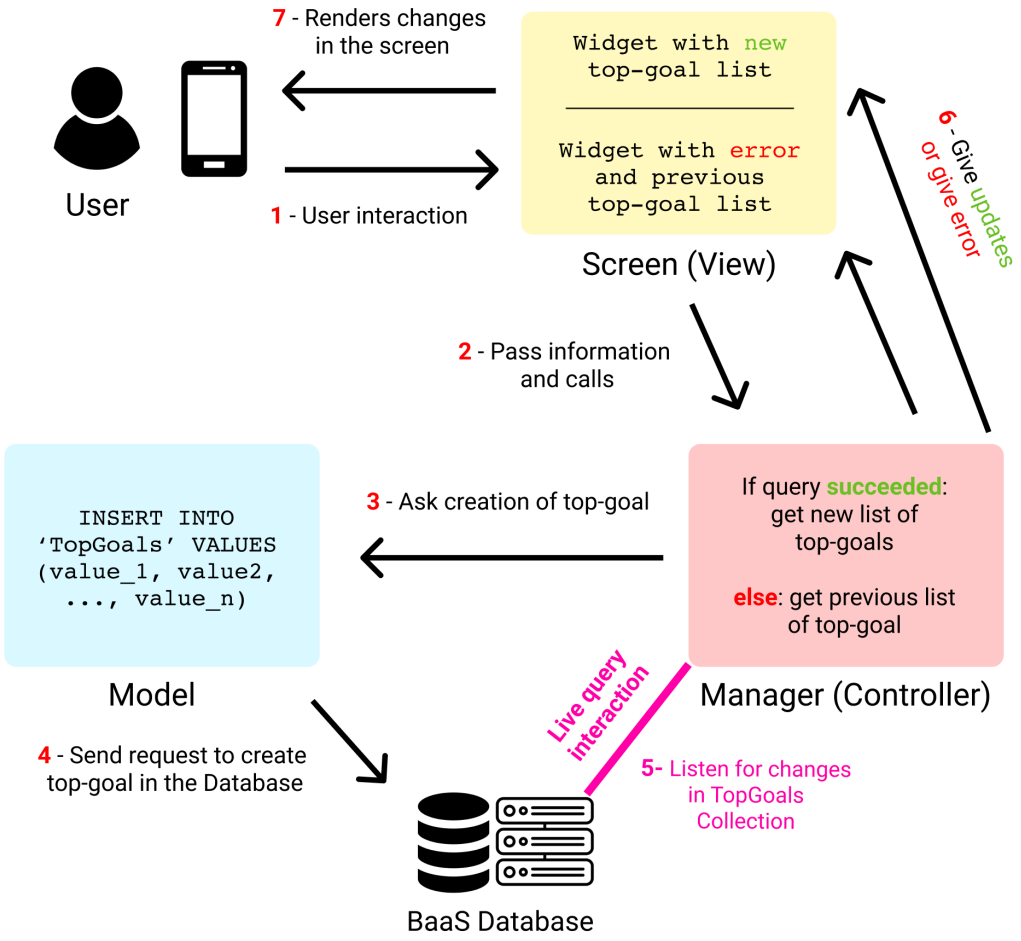


Figure 3.2: Red Nuclear Monkey's design model with a real use case of Alacrity.

3.2.2 Flow between main screens

The goal of this paper is to find the most relevant BaaS solution for a company with limited means and resources for the development of a mobile application. Therefore, the company knew beforehand that it would try several BaaS solutions within the same Flutter application. Then, it was necessary to have a stable and identical application base for each BaaS tested, thus avoiding wasting time. Subsequently, the idea was to have the same code for the application to perform the integration of BaaS more promptly and then only change the implementation of functions using the features of Backend-as-a-Service or any other data provider and API. The company therefore used the software reuse process, also known as code reuse. Code reuse is

the process of reusing code from old software developed to create a new software [22]. Considering the case of Alacrity, the team development reuses every time the entire application, which not includes the features implementations of BaaS. This process is one of the most well-known and used forms of object-oriented reusability [41]. Code reuse improves development efficiency and productivity [8][17] because it requires less implementation if the new software reuse previous functions. Besides, the new software achieves high reliability easily if it contains parts of previously tested and approved code [17]. Thus, respecting the behaviour of the application for each integration and testing of new BaaS solutions is essential. New BaaS implementation can not change this behaviour under any circumstances, otherwise this new implementation nullifies the benefits of reuse code and delays the development. Before explaining the application flow in detail, it is crucial to define the approach used for state management in Alacrity.

As explained in Chapter 2, Flutter rebuilds the user interface when an application state changes. Alacrity is an application including several states shared between several parts of the app and screens. Information for user authentication, user preferences, or like/unlike a comment are examples of application state. Therefore, it is imperative to manage these states to share them efficiently and to keep their values across the various screens. According to the official Flutter documentation[1], there are several ways to manage app states:

- **Provider**: The recommended approach, especially for live data. This chapter includes details about the Provider method.
- **setState**: The low-level approach to use for widget-specific, ephemeral state. Ephemeral state, also called UI State or local state, is a state contained in a single widget and not shared between other parts of the application. Ephemeral state do not use state management technique, but only a StatefulWidget.
- **InheritedWidget and InheritedModel**: A low-level approach used to communicate between ancestors and children in the widget tree. Providers and other approaches use InheritedWidget and InheritedModel.
- **Redux**: A popular state container approach in web application development.

[1]List of state management approaches for Flutter `https://flutter.dev/docs/development/data-and-backend/state-mgmt/options`

- **BLoC / Rx**: A family of stream/observable based patterns.
- **MobX**: A popular library based on observables and reactions.

This documentation is complete and offers examples of use for each of the methods presented. However, the documentation suggests choosing the state management method according to the complexity and nature of the application developed and the experience of the development team. Finally, it suggests using the Provider approach if the development team does not have a specific choice for two principal reasons. First, it requires little code. And then, the Provider approach uses concepts applicable in all other methods, which makes it possible to understand the basics of state management if Flutter is new to the development team. Thus, for state management inside Alacrity's project, Red Nuclear Monkey selected the Provider approach. This paper does not provide details for the other management states methods.

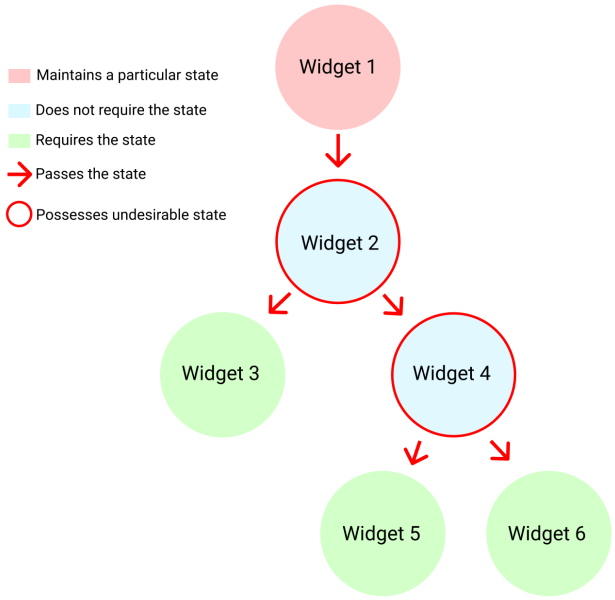


Figure 3.3: Illustration of an incorrect way to manage state.

The Provider and other approaches facilitate the sharing of states between the different widgets. Undeniably, it is possible to share these states without using these techniques, but developers have to pass the data explicitly to the widgets that need these states to operate. Explicitly transferring

this data from one widget to another can be troublesome as the application scales. Depending on the widget three, some child widgets may require specific states to use it while their parent widgets do not. Figure 3.3 illustrates this problem. Widget number 1 maintains a particular state. Widgets 3,5 and 6 need to have access to the state of widget number 1. Without using state management, you must pass the state to the widget number 1, 2 and 4 so they can transfer the state to the widgets 3, 5 and 6. This process is conceivable, but it is not an appropriate development approach since the widgets number 2 and 4 will have data they do not require. In that case, these widgets serve only as relays.

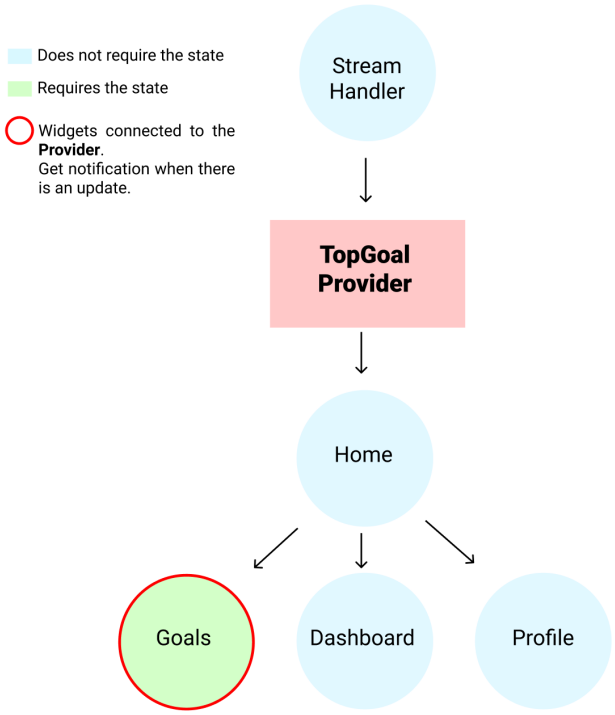


Figure 3.4: State Management using Provider.

Regarding the case of Alacrity, using the Provider tackles this problem. The provider is inserted in widget three and is connected to a data class. It allows all child widgets, i.e. those below it in the widget tree, to ascend the tree and access the data class, which is, therefore, the only instance of that class. When creating a project, Flutter does not directly provide the functionalities of the Provider. To use the features, the developers need to

install the Provider package[2].

Figure 3.4 illustrates a case of a provider present in the flow of Alacrity. When a widget changes a value of the data class associated with the Provider, all the widgets using the data of these classes get a notification. Therefore, this notification triggers a rebuild of the widgets to include the latest information and modifications. The Provider contains 3 concepts[3]:

1. **ChangeNotifier**: A class that allows providing information and notifications to listeners, e.g. to widgets in our case. This concept is rather an Observable. The unique feature of this class is the use of the `notifyListeners()` function, which allows listeners to receive a notification when a change has taken place. Below, an example of the model class of users of the Alacrity application. This class extends the `ChangeNotifier` class to access the `notifyListeners()` function. Whenever there is a change concerning the data of a user, there is a call of the `notififyLiserners()` function. The thesis presents the explanation of the implementation of the `onStreamUpdate()` function in Chapter 5.

Listing 3.1: Example of ChangeNotifier

```
class User with ChangeNotifier {
  User({this.id, this.email, this.name})
  ...
  User onStreamUpdate(DocumentSnapshot data) {
    ...
    notifyListeners();
    return this;
  }
```

1. **ChangeNotifierProvider**: A widget that provides the instance of the `ChangeNotifier` class to its descendants. In widget three, this widget is on top of widgets that need to have access to the data of the `ChangeNotifier` class. Multiple `ChangeNotifierProviders` can be inserted in the same places. This is one of the cases of Alacrity. The example below illustrates this case. In this situation, the `ChangeNotifierProvider` is provided to all widgets below the `StreamHandlerWidget` widget. However, developers must use a `MultiProvider` widget that contains multiple `ChangeNotifierProviders`, as shown in the example below:

[2]Provider package `https://pub.dev/packages/provider`

[3]Provider documentation `https://flutter.dev/docs/development/data-and-backend/state-mgmt/simple`

Listing 3.2: Example of ChangeNotifierProvider

```
class StreamHandlerWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    printer.i('Building wrapper');
    final AuthService authService = Provider.of<AuthService
       >(context);

    if (authService.isAuthorized) {
      return MultiProvider(providers: <SingleChildWidget>[
        ChangeNotifierProvider<User>(
            create: (_) => User(
                id: authService.authId,
                email: authService.authEmail,
                name: authService.pseudonym)),
        ChangeNotifierProvider<TopGoalManager>(
            create: (_) => TopGoalManager(userId:
               authService.authId)),
        ChangeNotifierProvider<SessionManager>(
            create: (_) => SessionManager(userId:
               authService.authId))
      ], child: Home());
    } else {
      return Authenticate();
    }
  }
}
```

Thus, all widgets below `Home`, and including this one, will have access to the instance of `User`, `TopGoalManager` and `SessionManager` classes.**Consumer/Provider.of**: The `Consumer` widget or `Provider.of` allow using the instance of the class provided by the `ChangeNotifierProvider`. Regarding `Provider.of`, the `listen:false` parameter can be used to not trigger the rebuild of a widget when calling the `notifyListeners()` function. Alacrity uses `Provider.of`.

Listing 3.3: Example of Provider.of

```
Widget build(BuildContext context) {
 final TopGoalManager tpManager = Provider.of<
    TopGoalManager>(context);
 ...
}
```

The example above is the widget responsible to display a list of a user's top-goals. `Provider.of<TopGoalManager>(context)` allows to

retrieve the instance of the class provided by the `ChangeNotifierProvider<TopGoalManager>`.

1.

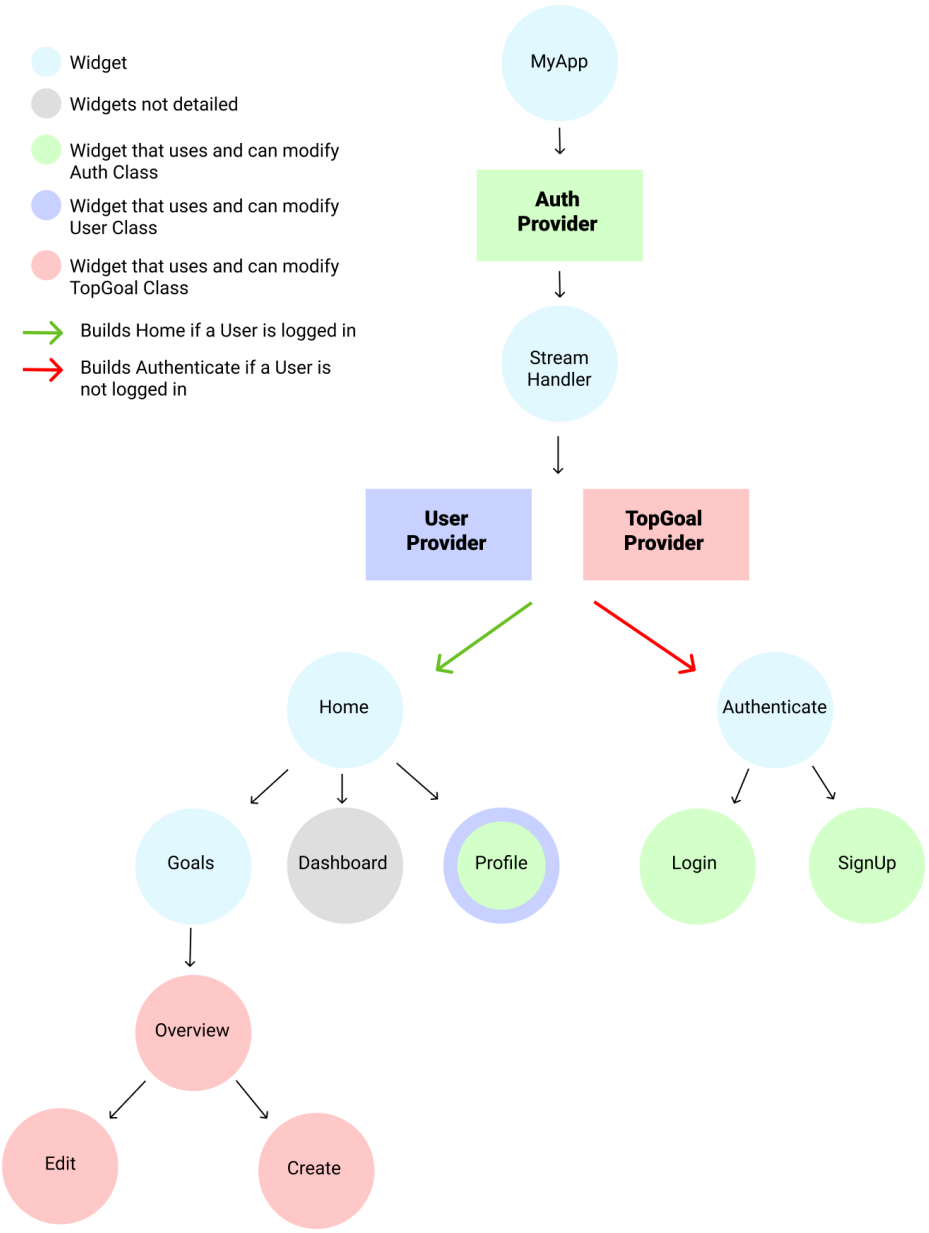


Figure 3.5: Flow of Alacrity between the main screens.

Thus, after defining the state management method used during the development of the Alacrity application, Figure 3.5 presents the flow of the applica-

tion between the primary screens presented previously in this Chapter. This part of applications has twelve central widgets in total. First, the MyApp widget runs the rest of the application and therefore runs the StreamHandlerWidget Widget by providing the Auth Provider. This Provider gives the authentication details of a user, for instance, it tells if a user has logged to the application on the current mobile device. Then StreamHandlerWidget builds:

- The Home widget if there is a logged user. In that case, the StreamHandlerWidget also provides the User Provider, which retrieves the user's information and the TopGoal Provider, which gives the information of a user's top-goals.
- The Authenticate widget if there is no logged user.

All widgets below Home can consume User, TopGoal and Auth data using the corresponding Provider. Therefore, each time a user changes the data of the User or TopGoal class, the Provider gets a notification, and it will rebuild the widget with the most recent data. When the user logs out in Profile, there are several changes in the Auth class. There is no more a logged user on the application. Finally, the StreamHandlerWidget widget receives this change about the user and builds the Authenticate widget, which returns the user to an authentication screen.

The next sections provide details on the Authentication, Goals, and Profile screens. These sections present the different widgets used to form the user interface and the interactions that the user experiences with these different screens. To illustrate these interactions, Red Nuclear Monkey used a category of Unified Modeling Language (UML): the use cases diagrams. UML is an industry-standard graphical language that simplifies the complex process of software designs [3]. This language uses graphical notations to express the object-oriented (OO) analysis and the design of software projects [10]. Two categories of diagrams represent these graphic notations:

- **Behavioural UML diagrams**, which describe the behaviour of a system, such as use case diagram, activity diagram, sequence diagram and state diagram.
- **Structural UML diagrams**, which analyze the structure of a system, such as class diagram, package diagram and object diagram.

Regarding the Alacrity project, the development team decided to only use the use case diagrams. Use cases capture the needs and requirements of a

user and a software system and their interactions [2]. Since Alacrity is a simple application, use case diagrams are sufficient to understand how the user interface works and how to communicate between different objects.

Note: To simplify the explanations, the presented user interface in the next sections is not the complete and final UI of the application.

3.3 Authentication screen

As shown in Figure 3.5 about the application flow, the authentication screen has two main widgets, the LogIn and the SignUp. Those widgets establish two new screens/views. The authentication screen represents only one of these two views, and a user can navigate easily through them. To experience the content of the Alacrity application, the user must log into it. Then, two cases stand out for a user:

- Either he is a former user, i.e. he is already registered. This user can log in via a LogIn screen or the SignUp screen, including different authentication systems such as Facebook and Apple ID,
- Either he is a new user, i.e. he is not registered. This new customer can create an account by registering through the SignUp screen or an authentication system integrated on this page.

The registration of a user is necessary to store their data in the database and thus be able to retrieve them during their next connections to the application. Therefore, these different widgets use the authentication and database functionalities of Backend-as-a-Service described in Chapter 5 of this paper.

3.3.1 UML: use case diagram

Figure 3.6 describes the interactions between a user and screen authentication. This schema shows the different actions of the user in 3 systems:

- one global system, representing the Authentication screen,
- and two systems integrated into the global system, i.e. the SignUp screen and the LogIn screen.

The application does not and can not show both SignUp screen and LogIn screen at the same time. The Authentication screen displays depending on the user's choice, either the SignUp screen or the LogIn screen.

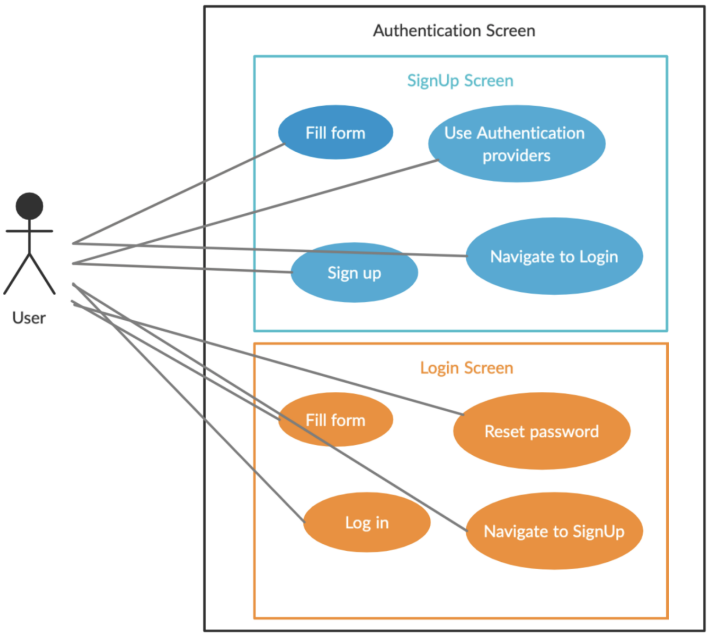


Figure 3.6: Use case diagram of Authentication Screen: User point of view.

3.3.2 Fundamental widgets

Three files compose the authentication screen:

- **authenticate.dart**: Provide the link between the LogIn screen and the SignUp screen,
- **signUp.dart**: SignUp view of the application
- **logIn.dart**: LogIn view of the application

The **authenticate.dart** file builds the LogIn or SignUp view based on a user's choice. There are several ways to perform this switch, such as the use of the Navigator functionality provided by Flutter or the use of states. This thesis presents these two approaches: this section shows the state approach and the Goals screen section illustrates the Navigator approach.

Authenticate contains only a state represented by a boolean. Based on this boolean, the Authenticate file builds the SignUp or LogIn view. By default, it initializes the boolean to true. When the boolean is true, the user faces the SignUp view, otherwise, i.e. when it is false, the user gets the LogIn view. Authenticate file also provides a function to change the value of

this boolean. SignUp and LogIn view receive this function. Thus, each view can execute this function and then directly affect the value of the boolean present in authenticate.dart, which will trigger the build of the other view. The code below presents this function and the approach to send it to the other files. This code also presents a part of the LogIn view which illustrates the constructor of this class, which has a function attribute to receive and use the function sent by Authenticate.

Listing 3.4: Content of the authenticate.dart file

```
bool showSignUp = true;

  void changeView() {
    setState(() => showSignUp = !showSignUp);
  }

  @override
  Widget build(BuildContext context) {
    return showSignUp
        ? SignUp(changeView: changeView)
        : LogIn(changeView: changeView);
  }
```

Listing 3.5: Constructor of the LogIn file

```
class LogIn extends StatefulWidget {
  const LogIn({this.changeView});

  final Function changeView;
  ...
}
```

The **signUp.dart** file contains the SignUp view, responsible for registering a user. The screen contains in order:

1. An Application Bar, represented by an AppBar widget and customized by Red Nuclear Monkey. This widget is present in each screen of the application. This Application Bar here only displays the title of the page, i.e. Sign Up.

2. Clickable buttons symbolized with RaisedButton widgets. These buttons allow the user to authenticate himself with Facebook or Apple ID. The SignUp view does not build the RaisedButton widget responsible for the authentication using Apple ID for Android and for OS versions that do not support this Apple feature. To manage the rendering of this button or not, the view uses a function present in the package responsible for adding the functionality of authentication with the Apple

ID. This function checks whether the mobile device is running on the Android or iOS environment. If it is running on iOS, it also checks the version of the operating system.

3. A Form Widget, including several TextFormField widgets. The Form and the TextFormField widgets check efficiently if the information that the user provided is valid via a global key linked to the Form and validators. TextFormField widgets take user inputs and save them in string variables, e.g. name, password, verificationPassword, email. Each TextFormField has a validator, as shown in the example below:

Listing 3.6: Example of TextFormField validator

```
TextFormField(
    ...
    validator: (String value) {
        if (value.isEmpty) {
            return 'Please enter a password';
        } else if (value != password) {
            return 'The 2 passwords do not match!';
        }
        return null;
    },
)
```

Thus, no TextFormField can be empty. Otherwise, an alert message warns the user that some fields are missing. In the example above, the TextFormField's validator of the 'Repeat Password' verifies that the password matches the same value the user entered for the 'Password' TextFormField.

4. A SignUp button, symbolized by a RaisedButton Widget, responsible for carrying out the creation of an account via the functionalities of cloud providers, e.g. Backend-as-a-Service. The code executed by this button use the Form key. When the user clicks on the button, the application check thanks to the key form the state of all validators. If the states are incorrect, i.e. some TextFormFields do not pass their validators, the user faces error messages.

5. A clickable text represented by a Text widget and a GestureDetector widget. The GestureDetector Widget makes a widget clickable. When the user clicks on this text, it executes the change view function explained in the previous, and therefore allows the construction of the LogIn widget.

The **logIn.dart** file, which produces the LogIn screen, has a very similar composition to the **signUp.dart** file. It contains:

Figure 3.7: Authentication screens of Alacrity application.

1. The customized AppBar widget displaying the name of the screen, e.g. 'LogIn'.

2. A Form widget and two TextFormField widgets. These two TextFormField widgets retrieve a user's email and password in variables of type string.

3. A LogIn button, symbolized by a RaisedButton. Once the user fills the TextFormFields, he can click on the LogIn button, which will have the same behaviour as the SignUp button, i.e. check that the fields are valid before executing the LogIn function using BaaS functionalities.

4. Two clickable Text widgets, also formed with GestureDetector widgets. The first Text widget allows the user to return to the SignUp page. The second Text widget performs the password reset function.

Figure 3.7 shows the view of the SignUp and LogIn screen and their interaction. It summarizes the different widgets used and illustrates the error messages triggered by the validators of some TextFormFields.

3.4 Home Screen

The Home screen can display three widgets: Dashboard, Goals and Profile. The thesis gives details about two of them. Therefore, these three widgets form three new screens/views for the user. They are accessible through Home widgets. The Home widget allows you to display three categories, also called tabs, on the screen, while showing the content of one category. The user can click or swipe on any of these categories to display the content and access the functionalities of that tab. This thesis does not include any other details about this Home widget because it does not directly use Backend-as-a-Service features.

3.4.1 Goal Screen

The Overview widget produces the Goal screen. This view and this widget presents the list of a user's top-goals, the information about the current session, and allows the user to create a new top-goal. Thus, the Overview widget also allows the customer to navigate toward two different views and thus new widgets: Create and Edit. These two widgets respectively provide the user to create new goals and edit their current goals. The Edit View is accessible when a user is in the Overview view and clicks on an existing

top-goal. This interaction triggers the construction of the Edit widget. The Create View is available when the user clicks on a specific card to create a new top-goal. At the end of the modification or the creation of a top-goal, the application rebuilds the Overview widget and therefore redirects the user to their list of top-goals. These three views available to the user use the Backend-as-a-Service database as they display sensitive data to the customer, e.g. top-goal information in this specific case.

3.4.1.1 UML: use case diagram

Figure 3.8 describes the interactions between the current user and the various screens forming the Goal view. The schema has three parts, representing the three different views that the user can have while browsing the Goal category.

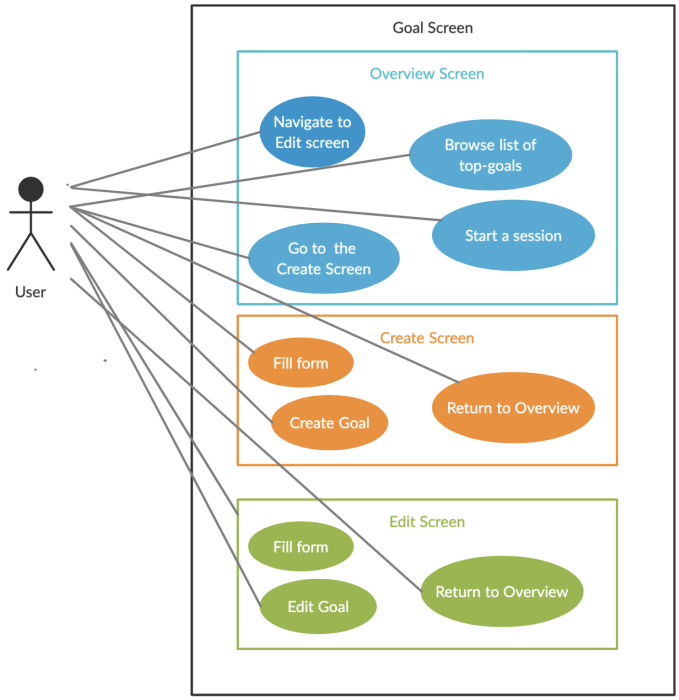


Figure 3.8: Use case diagram of Goal Screen: User point of view.

3.4.1.2 Fundamental widgets

Three files also compose the Goals screen:

- **overview.dart**: Allows the user to follow the progress of a top-goal and go to the goal creation screen.
- **create.dart**: Provides the user to create a goal.
- **edit.dart**: Grants the user to edit a top-goal.

*Note: The **overview.dart** file also displays information about sessions. However, this thesis does not give details about them. These three files form three new views available to a user.*

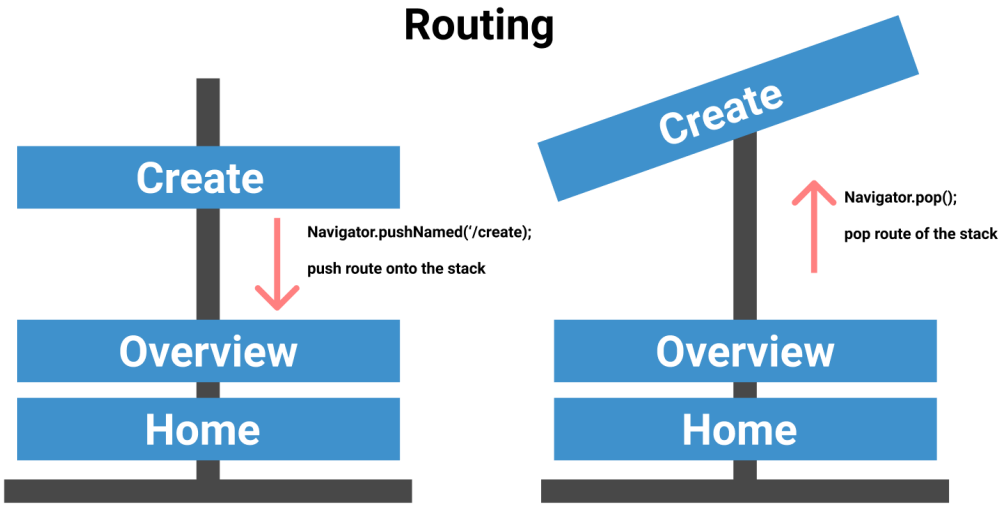


Figure 3.9: Routing example using Alacrity use case.

To make this link among these three different screens, the Alacrity project uses Routing, including the Navigation functions in the same file: **router.dart**. Compared to the state approach, presented with the authentication screen, this approach makes the navigation and data transmission more efficient between multiple views. The **router.dart** file that implements this approach contains a Navigator widget to reference all the routes for these different views. Each route is used to display a new view to the user and to pass functions and arguments as parameters to the files used to build these views. Thus, the **router.dart** file also contains all the functions to navigate from one view to another, using the functions of Navigators, such as `Navigator.pushNamed` and `Navigator.pop`. `Navigator.pushNamed` allow the application to show a new view by providing the name of the route,

and `Navigator.pop` deletes that view and return to the previous view. The application acts as a stack of screens and the Navigator functions add or remove views from this stack. Overview is the initial route for the Goal screen. Figure 3.9 illustrates this idea of a stack.

The **overview.dart** file contains:

1. The functions provided by the Router which allow navigation to the Create and Edit view.

2. The TopGoalManager Provider, which allows him to retrieve the active top-goal of a user

3. The custom AppBar widget, displaying the page title, e.g. 'My Top Goals' and a subtitle, showing the number of top-goals already completed by the user.

4. Three Section widget. The first section is 'Quick Actions'. This section consists of a ListView widget that lets you make a list of widgets scrollable. This listing features custom cards using the BoxDecoration and Text widgets. If a user does not have an active top-goal, a 'Create new top-goal' card is visible, otherwise, this card disappears. This case implies that a user can only have one top-goal active at a time.

   The second section is 'Progression'. This section only contains a card that gives information about the user's active top-goal. The user can click on this map and navigate to the Edit view.

   Finally, the last section 'History' presents the history of top-goals also using the ListView widget.

The **create.dart** file contains:

1. A function provided by the Router file to return to the Overview view.

2. The User Provider, to retrieve the identifier of a user and the TopGoalManager Provider to call for the creation of a top-goal.

3. The custom AppBar widget, indicating the title of the page, e.g. 'New Top Goal' and a subtitle, indicating the number of the top-goal.

4. Four Section widget. The first section 'General' consists of a Text widget explaining the necessary and general information for creating a top-goal. Two InputText widgets allow you to retrieve the user's inputs to retrieve the name of their top-goal and the number of hours they

want to spend dealing with their goal. A DatePicker widget is used to retrieve the user's goal deadline. Finally, a Text widget displays the minimum number of two hours sessions the user needs to complete reaching their goal.

The second section is 'Motives'. This section first includes a Text widget that tells the user to fill in at least 3 motivations to complete their goal. Then the user can enter his motivations using an InputText widget. These motivations appear in a list, thanks to the use of the ListView widget. Card widgets illustrate these motivations. The user can also click on one of their motivations to remove it from the list.

The third section 'To-dont's' is similar to the second section. However, the user does not need to enter at least 3 To-dont's.

The fourth 'Summary' section comprises a Text widget explaining the summary of the top-goal the user is about to create. Finally, this section includes a RaisedButton widget to create a top-goal for this user and return to the Overview view. If the user has not entered a Title, planned hours and motivations, the summary is not accessible. Instead, the user faces a text informing him to fill the required fields.

The **edit.dart** file does not contain the latest UI changes of Alacrity. Therefore, this view is particularly simplified, and its composition is very similar to the Create view. This file contains:

1. A function provided by the Router file to return to the Overview view.
2. The identifier of the top-goal that needs modifications, also provided by the file Router.
3. The TopGoalManager Provider, to retrieve the entire top-goal thanks to its identifier.
4. The custom AppBar widget, specifying the name of the top-goal that the user changes.
5. Two Sections widgets. The first section 'Motivation' includes the list of motivations associated with the top-goal and presented with the ListView and Card widgets. The user can delete one or more motivations or add new motivations to this list using an InputText.

   The second section 'To-dont's' has the same behaviour, but with the 'To-dont's' associated with the current top-goal.

6. Two RaisedButton widgets, one to save the new values of the top-goal and another to delete it. These two buttons use the functionalities of a cloud provider.

Figure 3.12 presents the different views of Goal created by these three files and their interactions with each other.

3.4.2 Profile Screen

Finally, the Profile screen, also accessible via the Home widget, allows the user to log out and have his information such as his name, his first name and his picture. Shortly, Alacrity will allow users to edit their information and upload a profile picture. This widget only uses the Backend-as-a-Service database functionality to deliver and make available the information of the current user.

3.4.2.1 UML: use case diagram

Figure 3.10 shows the user interactions on this screen. This diagram also shows the next features and therefore the future interactions. The schema illustrates these future interactions by the purple colour.

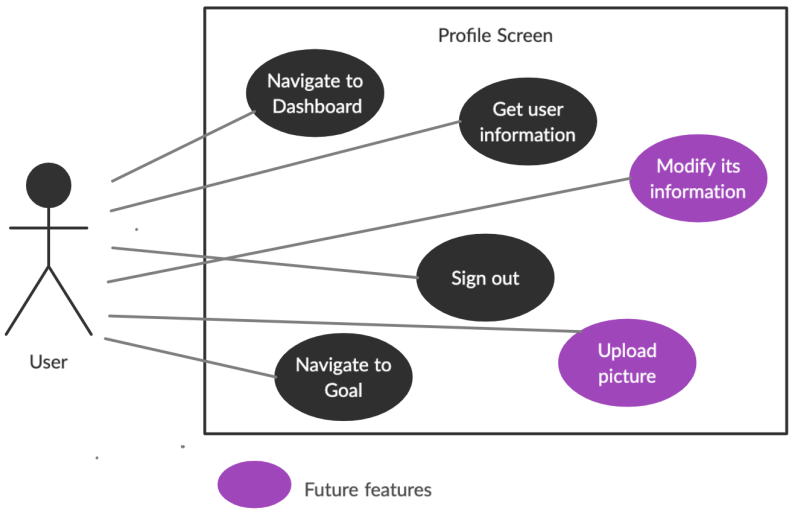


Figure 3.10: Use case diagram of Profile Screen: User point of view.

3.4.2.2 Fundamental widgets

Since the current version of the application does not yet allow changing user information, the composition of the Profile screen is highly elementary. The **profile.dart** file contains:

1. The user Provider to retrieve information related to the current user.
2. The AppBar widget, showing the page title and a subtitle revealing how long the user has been a registered member of Alacrity.
3. A Section widgets composed of four Rext widgets that illustrate the following information about a user: his User Unique Identifier (UID), his name, his email and his number of points.
4. A SignOut button, illustrated by a RaisedButton widget. This button allows the user to log out. When the user clicks on this button, it executes the log out function implemented using the functionalities of a Backend-as-a-Service.

Figure 3.11 shows the current version of a user's page. This view does not include the final Alacrity UI.

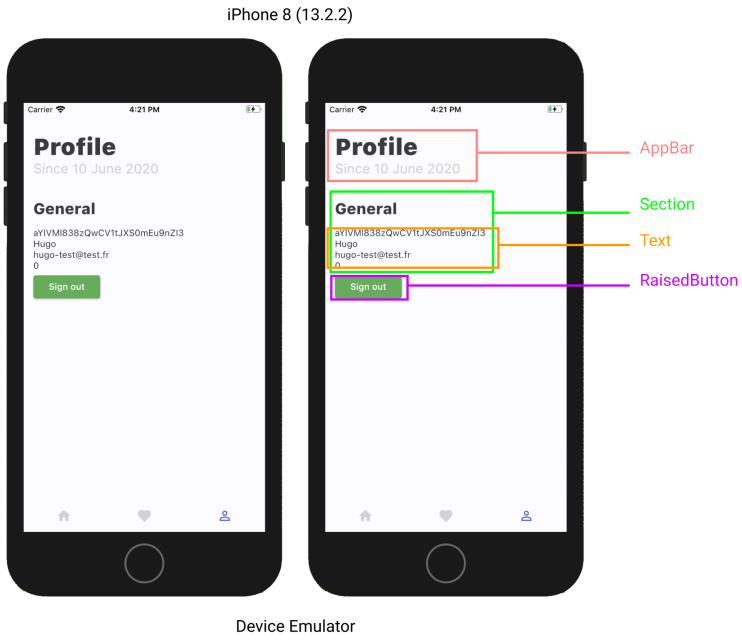


Figure 3.11: Profile screen and widget composition.

3.5 Required features of MBaaS

After implementing the user interface for the screens presented in the previous sections, the choice of Backend-as-a-Service features is accessible to make. The application needs the authentication features, including traditional authentication of a user who registers with an email and password and with authentication systems such as Apple ID and Facebook. Besides, the application requires a database to store all the data necessary for the operation of the application, i.e. user and top-goals information.

Red Nuclear Monkey also suggested for this thesis to use the Push Notification feature. As the company has limited resources, the company has been encouraged the use of a Backend-as-a-Service that can conveniently be integrated into a Flutter project. The company did not know in advance which BaaS could achieve this approach. However, the company suggested for this research to look at the services offered by Amazon AWS Amplify, Microsoft Azure, and Firebase. The different functionalities and the restrictions imposed by the company Red Nuclear Monkey serve as selection criteria to choose the most suitable BaaS for developing the Alacrity application. The next chapter summarizes the Backend-as-a-Service solution and the selection of the BaaS retained for the development of Alacrity.

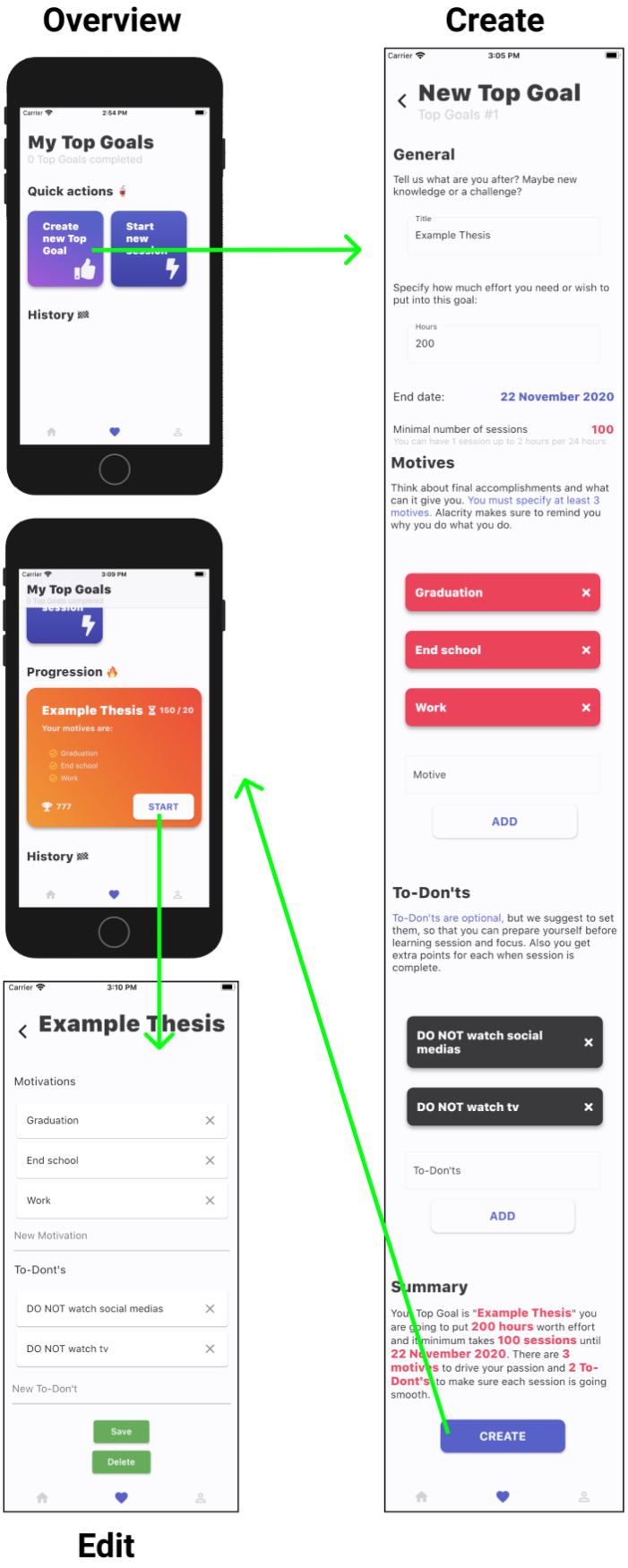


Figure 3.12: Presentation of the three Goals screen and their interactions.

Chapter 4

Backend-as-a-Service

This chapter presents the notion of Backend-as-a-Service and the choice of BaaS for the development of Alacrity according to the criteria of the company Red Nuclear Monkey.

4.1 Backend in mobile/web development

First, before defining the notion of Backend-as-a-Service, it is essential to understand what the backend represents in the development of web and mobile applications. To understand the purpose of a backend, it is necessary to mention what the frontend is and their relation. The frontend, also known as the client-side, represents what a user can interact with and see on a mobile application or a web page. For example, Chapter 3 presents the frontend of the Alacrity application, especially the UI and interactions that the user can perform. The backend, or server-side, represents everything that happens on the server and database side of the web or mobile application. The user is not aware about the backend and does not directly interact with it. However, the backend is responsible for what happens during interactions between a user and the application frontend. The server-side application interacts with the database through an API to create, retrieve or change the data. Figure 4.1 shows an example of the interaction between the frontend and the backend of a mobile application.

1. First, the user interacts with the Alacrity login page by filling in the authentication fields to log in. When the user clicks on 'Login', the application server-side verifies that the fields are correct.

2. Once the verification is done, the server-side sends the data to the database, to check if the user exists.

3. If the user exists, the database returns the information to the server-side. If the user does not exist, the database sends no information back.

4. The server-side can then retrieve the data needed, such as information about a user, and send it back to the client-side, which will allow the user to access the new data.

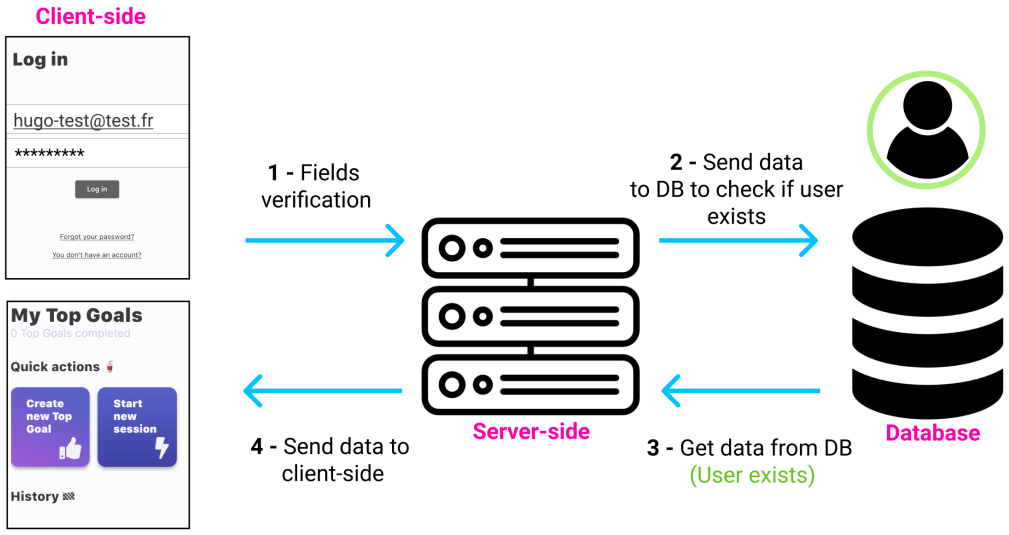


Figure 4.1: Example of backend and frontend interaction for a mobile application.

Hence, the backend is responsible for all the functionalities of an application and for the interaction with servers and databases. Nowadays, there are at least two types of backend: custom backend and backend-as-a-service. These two types of backend have a unique approach, and they both present benefits and disadvantages. The custom backend is a backend created specifically for an application. This backend is developed internally by companies and includes all the functionalities necessary for the application, such as user authentication, data management or notifications. The cost and development time are significant for this approach. Undoubtedly, companies require a development team to build custom backend. Therefore, the custom backend is not a suitable solution for companies that do not want to waste time developing a custom backend for each new application and for small companies with limited resources and employees. Thus, these companies are turning to the other backend solution: Backend-as-a-Service.

4.2 Backend-as-a-Service

Backend-as-a-Service (BaaS), also called Mobile-Backend-as-a-Service (MBaaS), are platforms that allow the use of pre-developed backend stored in the cloud. These platforms provide a way to connect web or mobile applications to backend cloud storage, allowing developers to use the most popular features of applications such as file storage, authentication, cloud data, social media integration or analytics. These features are accessible through the use of SDKs and APIs like Representational state transfer (REST) API [1]. This BaaS solution allows companies to avoid having and maintaining specific servers for the backend. Hence, this approach considerably reduces the cost and the development time compared to the custom backend. Therefore, BaaS solutions allow developers to focus more on the user experience and on the frontend and business logic of the application. The BaaS model provided by cloud providers includes the functionalities of the Infrastructure-as-a-service (IaaS) and Platform-as-a-Service (PaaS) models. Indeed, the PaaS model is not sufficient and does not provide the functionalities that web and mobile developers are looking for. BaaS platforms abstract away the complexities of launching and the management of an infrastructure, while also including the useful resources that developers need to speed-up their web or mobile applications [21]. The functionalities provided by these different models are:

- **IaaS**: Provide data center, servers, storage and networking
- **Paas**: Provide IaaS functionalities, deployment, management and scalability
- **BaaS**: Provide PaaS functionalities and features to Build Backend using SDKs and APIs. For instance, according to a cloud provider offering a BaaS solution[1], the most used BaaS features by developers are:
    1. Scalable Database
    2. APIs (REST and GraphQL)
    3. Business Logic via Cloud Code Functions
    4. User authentication
    5. Social Integration (Facebook, LinkedIn, Twitter, etc.)
    6. Email Verification

[1]Backend as a Service - What is a BaaS? - Back4App `https://blog.back4app.com/backend-as-a-service-baas/`

7. Push Notifications
8. Geolocation

The BaaS model remains between the PaaS and Software-as-a-Service (SaaS) models. Figure 4.2 illustrates the positioning of these different cloud services models according to the functionalities they provide. The more functionalities a model provides, the higher it is in the model hierarchy. Besides, this figure summarizes the composition of Backend-as-a-Service.

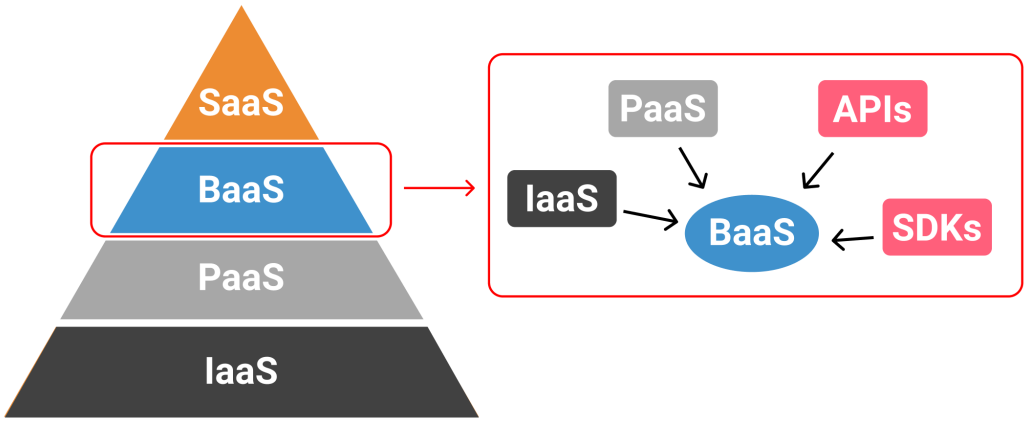


Figure 4.2: Cloud services models and composition of BaaS.

Consequently, BaaS and custom backend solutions are two distinctive approaches, and companies use them according to their technical and economic criteria. The next section presents the technical and business indicators that are decisive for the choice between Backend-as-a-Service and custom backend.

4.3 Important indicators for choosing BaaS approach

As seen in the previous section, BaaS are solutions for including easily backend features widely used by developers when developing web and mobile applications. However, *what are the technical and economic factors that make a company choose the BaaS approach over a custom backend?*

4.3.1 Main business factors

There are several business advantages using BaaS solutions. One of the most decisive benefits is **development speed**, which allows companies to save development time and thus increase productivity [13]. The development of an application takes a considerable time. Developing a specific backend for an application inevitably increases this amount of time. Indeed, the more features an application contains, the more time developers take to develop it. Thus, the services and features offered by BaaS save considerable time since the features are already developed and ready to use.

Another advantage of BaaS is to **reduce the time to market** for companies. As Backend-as-a-Service solutions reduce the development time, logically, the developed application ends up on the market much faster [13]. Therefore, companies can try their products or services in the market earlier. This advantage of BaaS is very beneficial for small companies and startups since the primary challenge for these businesses is to find a market-fit product. Hence, these companies will be able to try out several ideas and test their potential products while reducing the time of development and testing. Besides, with a reduced market time, BaaS seems to be an ideal solution for creating an MVP, since, for an MVP, it is crucial to cut the time to market.

Finally, **the cost** is one of the principal advantages of Backend-as-a-Service [31]. Developing an application from scratch can be very expensive, especially if it has many features and services such as security or backups. BaaS solutions offer these services at an affordable price and businesses do not need to pay for each service individually, unlike a custom backend. Even though BaaS services are chargeable, it is cheaper to use them than to develop a custom backend for businesses [37]. Also, BaaS platforms provide a serverless architecture for the company, which means users do not need servers on their side [13]. This architecture saves companies from buying servers to run their application on, which saves a subsequent amount of money. Besides, the backend developers team can be smaller, which reduces the payroll of a company. Finally, businesses benefit from 'pay-as-you-go', which means they pay only for the resources needed to run the application and user traffic. VantageBP[2] and Food Cowboy[3] are two examples of businesses that have saved money by using BaaS. Regarding VantageBP, the company saved $500,000 over one year and a half, by reducing the number of backend developers while using the benefits of BaaS. For Food Cowboy, the company saved $200,000

[2]How VantageBP saved $500K USD using a BaaS - Back4App `https://blog.back4app.com/vantagebp-case/`

[3]The Food Cowboy Case - Back4App `https://blog.back4app.com/backend-as-a-service-baas/`

by developing their initial version of their project, i.e. MVP, with the help of a Backend-as-a-Service named Parse.

4.3.2 Main technical factors

Backend-as-a-Service also allows companies to benefit from technical advantages for their development and application. As explained previously in this chapter, with the use of BaaS features, developers will spend less time developing the backend and thus **focus more on frontend development** [31]. This gain of time allows developers to spend more time and effort building an excellent user experience, which is essential for any application. The user experience is one compelling principle in the judgment of an application for a customer. If the user experience feels positive, the customer has a decent chance to be satisfied and therefore use the application.

Besides, BaaS allows developers to **code high-value lines of code**. They do not need to write boilerplate code and perform repetitive tasks since BaaS already provide pieces of code that are used to implement functionalities [13]. It also allows developers to focus on business logic coding.

Beyond providing the basic functionalities of a backend, BaaS solutions provide **several features 'ready to use'** [31] and which developers do not need to implement themselves, such as integrating social networks, user authentication and push notifications.

Finally, with BaaS, companies **do not have to maintain servers** and have **pre-configured backup procedures** in case of data problems.

Thus the economic and technical advantages seen above can be beneficial for both small companies or startups and well-established businesses. However, some disadvantages of BaaS can cause some companies to choose the custom backend approach.

4.3.3 Disadvantages of BaaS

According to a study conducted in 2015 by S. Rasthofer et al. [36], BaaS **do not offer optimal security**. These researchers have shown through their paper that with attack tools, it was possible to get sensitive customer data such as verified e-mail addresses, health records, complete employee and customer databases. If the application developed by a company includes sensitive customer data, S. Rasthofer et al. [36] recommend to not store this data in a shared server provided by a BaaS solution. However, nowadays companies probably store the data in the cloud, and trust cloud solutions by

paying options to have isolated storages and other security measures applied to their data.

Besides, BaaS solutions offer **low reliability**. Each dependent application developed using a BaaS highly depends on this BaaS platform and the features offered by this service. Therefore, if the BaaS platform shuts down for any reason, it affects the application which might stop working. Parse is a Backend-as-a-Service that illustrates this example where thousands of users had to find a new BaaS solution because Parse was no longer maintained by Facebook[4].

BaaS platforms also offer **low flexibility**. If an application requires unique, authentic or overly complex functionalities, the company itself will have to develop these functionalities as BaaS will probably not offer them [37].

Finally, BaaS seem to have **low scalability** [31] since BaaS services are seen as short-term solutions and are not intended for applications that expect to grow.

4.4 Popular Backend-as-a-Service solutions

This section presents a summary of some providers offering a Backend-as-a-Service solution. These cloud providers seem to be among the most popular providers to offer this service:

- **Firebase**: This BaaS platform was acquired by Google and aims quickly to help in the production of new applications. Like most BaaS, Firebase offers a free limited plan. Firebase has many features such as analytics, real-time databases, user authentication, push notifications, and hosting. Googles bases this Firebase platform on four pillars: Development Features, App Quality, Analytics and Growth. This paper offers details about these four pillars later in this chapter. The advantages of Firebase lie on the real-time database and the use of machine learning.

- **AWS Amplify**: AWS Amplify is a library provided by Amazon Web Services. It helps developers to create serverless applications for iOS, Android, React Native and web frontends. Like other BaaS, this service offers several services such as storage, APIs, analysis service, means of authentication for users and notifications. AWS Amplify offers a

[4]Facebook's decision to close Parse angers developers - BBC - `https://www.bbc.com/news/technology-35441445`

limited free tier and a Pay-as-you-go plan. This service is one of the most complete in terms of functionalities provided. Besides, several famous applications such as Periscope, Netflix and Airbnb use AWS Amplify as Backend-as-a-Service.

- **Parse**: Parse is an open-source framework with a large community, especially on GitHub. The goal of this Backend-as-a-Service is to create applications much faster. To use this framework, companies must use a hosting service or host the framework themselves. However, this service is free, which means companies only need to pay their servers to host the framework or to pay a hosting service such as Back4App. Parse has several advantages. Indeed, the framework is simple to use; the code provided is easily modifiable to adapt to the needs of the user; it is accessible for almost all platforms and allows to manipulate data easily with little code.

- **Azure Mobile Apps**: This Backend-as-a-Service solution is provided by Microsoft and integrates with Xaramim, a frontend cross-platform and Azure. It provides several mobile features like other BaaS, such as data storage, push notifications and authentication using Azure. Developers can create backend logic using Node.JS or C. Some advantages of Azure Mobile Apps are the security protocols in which Microsoft provides enterprise-level security to create apps, and AD Integrations, allowing enterprises to use corporate sign-on.

- **Kinvey**: This BaaS mainly focuses on businesses. It offers many features including mobilization for enterprise data, use of cloud-native services and locations services. It has a free version limited to 100 users. Beyond that, the entry price for using this service is $2,500/year, which is substantial, especially for small companies or startups. With this price, Kinvey focuses on companies that are already well established. Kinvey offers a 24/7 customer support and integrations such as Oracle, Salesforce and Microsoft Active Directory.

4.5 Backend-as-a-Service suitable for the company

This section presents the Backend-as-a-Service solutions selected for the development of the Alacrity application based on the criteria of the company.

First, Red Nuclear Monkey's criteria were to find a BaaS that could easily be integrated with Flutter, to keep development time and cost as low as possible. Then, the company suggested looking at AWS Amplify, Firebase, and Azure Mobile Apps since they are familiar platforms for Red Nuclear Monkey. To meet these expectations, the company carried research out to find BaaS easily integrable with Flutter. The development team analyzed each cloud provider from the list presented in the previous section to determine if these platforms provide support for Flutter. Besides, the team explored the Flutter's package sharing website[5] to find if libraries were available to use and integrate the functionalities of the different Backend-as-a-Services. As a result, only Firebase and Parse showed genuine support for Flutter. In fact, during the application development period, AWS Amplify, Kinvey, and Azure Mobile App did not officially offer integration for the Flutter framework.

A few packages developed by the Flutter community are available for some features of AWS Amplify, such as the authentication service. However, these packages had shown many issues[6] during the development time and the company could not consider these solutions as sustainable and reliable. Red Nuclear Monkey preferred direct support of cloud providers, i.e. packages developed by BaaS providers rather than independent developers. Further, no package appeared to offer AWS Amplify push notification and database service. However, this thesis and the Alacrity app required these two functionalities for the operation of the app and for the comparison of the different Backend-as-a-Service. The team development could have developed a REST API to access the AWS Amplify's database service, but developing such an API does not meet the company expectations as it adds development time and costs. Several developers requested official AWS Amplify support for Flutter[7] to benefit from the full functionalities of this service since Flutter gained in popularity. However, at the time of writing this research, AWS Amplify now supports the Flutter framework[8] and provides the functionalities needed for the Alacrity application.

Regarding Azure Mobile App and Kinvey, the package sharing website

[5]Packages for Flutter - `https://pub.dev/`

[6]Example of issue for an Amplify library - `https://github.com/agnostech/flutter_amplify/issues/11`

[7]Flutter support AWS Amplify - `https://github.com/aws-amplify/amplify-js/issues/1852`

[8]Announcing AWS Amplify Flutter - `https://aws.amazon.com/fr/blogs/mobile/announcing-aws-amplify-flutter-developer-preview/`

does not offer any package providing the features of these cloud providers. Still, these two BaaS platforms do not officially support Flutter.

Thus, **Firebase** and **Parse** are the two Backend-as-a-Service chosen to support the development of Alacrity. The Red Nuclear Monkey company decided not to host the Parse framework itself, but rather to use a hosting service. The company justifies this choice by the time savings that a hosting service can provide since it avoids having to have your own server to configure to initialize Parse and its configurations. The choice of this hosting service is **Back4App**. The next section offers more details regarding the two BaaS solutions chosen, and an overall comparison of these two services.

4.6 Global comparison: Firebase and Parse

As seen previously during the presentation of examples of Backend-as-a-Service solutions, Firebase is a BaaS platform launched in 2012 and acquired by Google in 2014. This platform provides features for developers to speed up cloud development in a web or mobile application. Google launched Firebase platform in 2016 to allow developers to generate income and thus increase their business [24]. Concerning Back4App, it is a platform for efficiently using a flexible and scalable backend based on Parse Platform. In the beginning, Parse was a Backend platform acquired by Facebook in 2013. In 2016, Facebook made Parse Open-Source and then in 2017 Facebook stopped maintaining Parse, leaving the Parse community to take over. To still use Parse, developers have to host Parse Server themselves or rely on hosting services, such as Back4App. This section presents the core features of Firebase and Back4App and a comparison of the different frameworks they support and their prices.

4.6.1 Features

Firebase is a service that offers many complete features. From 2016, Google has improved its service and offers a platform based on four pillars:

1. **Development**: This section contains all the features that developers want when deciding to use BaaS, such as authentication, database, cloud functions and hosting.

2. **Quality**: Google dedicates this section to the operation of the application, allowing users to get access to report crashes, observe per-

formance, perform tests and distribute the application to a group of testers.

3. **Analytics**: This part provides access to detailed statistics to measure and analyze user interactions with the developed application.

4. **Growth**: This part contains several additional features that can improve other features, such as Cloud-Messaging which allows sending a notification to targeted users, or Predictions which predict users' behaviour based on the analyzes of the application.

Regarding Back4App, the service provides less functionality than Firebase, but provides all the basic functionality of a Backend-as-a-Service for an application. The service offers the following principal functionalities: Relational Database Schema Queries, Cloud-Code Functions, Real-time queries, GraphQL REST APIs and Notifications.

4.6.2 Support

Officially, Back4App does not support Flutter, unlike Firebase. However, as Parse is Open-Source, the Parse community supports Flutter by having created a complete and detailed package for this framework[9]. Therefore, it is easy to use the Back4App platform with Flutter. According to their respective documentation, Firebase[10] and Back4App[11] both support the integration of their platform with Android, iOS and the web (including JavaScript and some frameworks like ReactJS or Angular).

4.6.3 Price

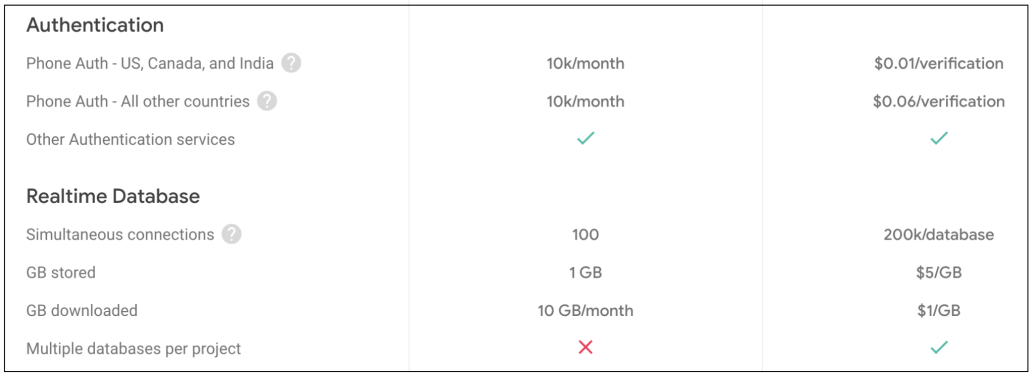
Regarding the price, the two platforms have different strategies. Firebase offers a free service and a Pay-as-you-go service. Figure 4.3 illustrates the Pay-as-you-go price for Authentication and Database features. Beyond 10,000 authentications, Firebase's Pay-as-you-go charges the company with $0.01 per verification for US, Canada and India countries and $0.06 per verification for other countries. Regarding the real-time database, the Pay-as-you-go plan allows you to get a greater number of simultaneous connections, several databases per project. From a stored GB, each new stored GB costs $5.

[9]Parse for Flutter - `https://pub.dev/packages/parse_server_sdk`

[10]Firebase Documentation - https://firebase.google.com/docs

[11]Back4App Documentation - `https://www.back4app.com/docs/get-started/backend-as-a-service`

Also, from 10 GB downloaded on the database, each new downloaded GB costs $1.



Authentication		
Phone Auth - US, Canada, and India	10k/month	$0.01/verification
Phone Auth - All other countries	10k/month	$0.06/verification
Other Authentication services	✓	✓
Realtime Database		
Simultaneous connections	100	200k/database
GB stored	1 GB	$5/GB
GB downloaded	10 GB/month	$1/GB
Multiple databases per project	✗	✓

Figure 4.3: Firebase Price for Authentication and Real-time Database.

Meanwhile, Back4App offers three plans:

1. Free: Ideal for developing, learning and prototyping
2. Shared: Launch an app quickly using their serverless architecture
3. Dedicated: Ideal for production apps that require dedicated infrastructure resources

Concerning Alacrity application, Red Nuclear Monkey would consider the 'Shared' option, since the company does not need a dedicated infrastructure for the application. Several options are available in the 'Shared' offer. Each of these options offer specificities more or less adapted to the needs of an application and its traffic. Figure 4.4 shows the different plans offered by Back4app considering the 'Shared' option. Back4App does not offer a concrete Pay-as-you-go plan since it charges the user monthly, even if the user has not used all the resources offered by the plan. However, with the Pay-as-you-go offer, the user can have additional chargeable resources if the user needs them.

Thus, the two commercial strategies of these platforms stand out. Back4App offers in addition to its free version, several paid versions which are monthly and potentially chargeable as needed by the user. In contrast, Firebase offers a Pay-as-you-go that only charges the users for the resources they need.

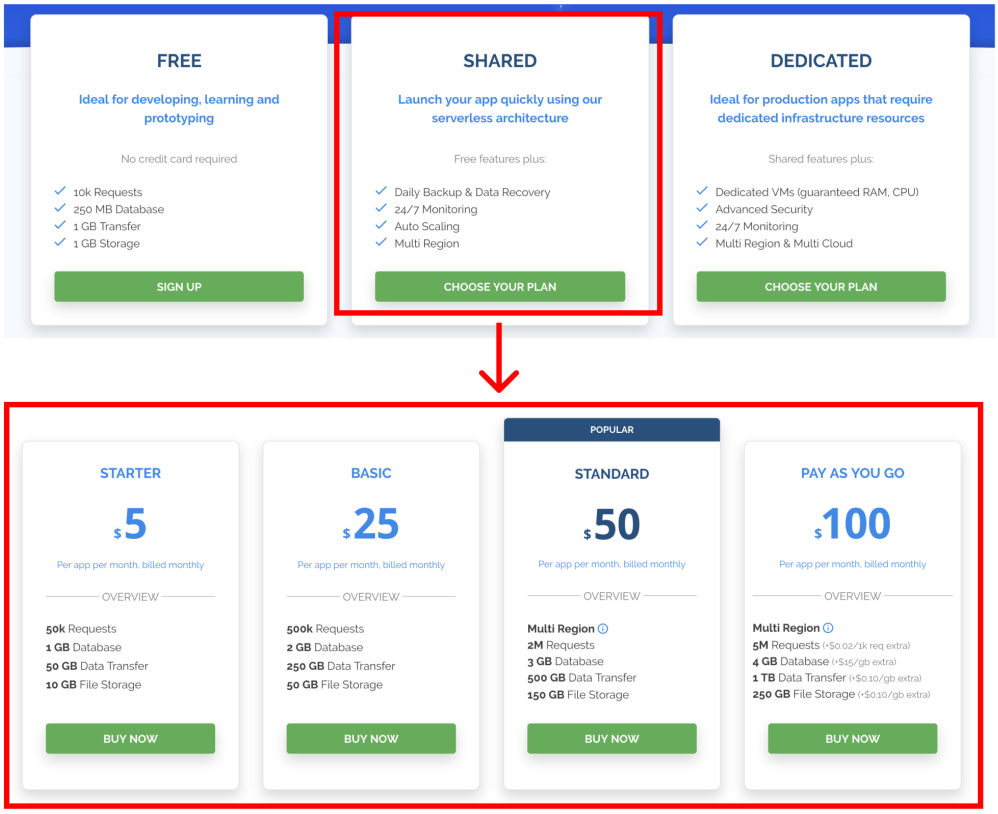


Figure 4.4: Back4App plans prices and Shared plans prices.

Chapter 5

Integration of BaaS functionalities

This chapter presents the installation of Back4App and Firebase solutions within the Alacrity project, and the implementation of the various functionalities required for the application. As a reminder, the different functionalities are: authentication, reset password, database and push notifications.

5.1 Development environment

This chapter offers samples of code, exposing the implementation of required features. To implement these features, the development team chose Android Studio as the integrated development environment (IDE) as Android Studio offers a complete, integrated IDE experience for Flutter. Android studio is the recommended IDE for Flutter in Flutter's documentation. The development team used version 4.0 of Android Studio and version 1.17.4 of the Flutter framework.

5.2 Setting up Backend-as-a-Service solution

To use the different functionalities that Back4App and Firebase offer, companies must first install these Backend-as-a-Service within their project. This section explains how to set-up Firebase and Back4App in a Flutter project assuming Android Studio and Flutter are already installed.

5.2.1 Setting up Firebase

First, the company needs to create a Firebase project through the Firebase console[1]. Once the project is created, the company must register the Android and iOS application in this Firebase project. To do this, the company must go to the Firebase project page and add an Android app and add an iOS app. When adding applications, the company must provide the package name of their applications. Then, the company can register their applications and then download and add the Firebase configuration file to use Firebase products.

- For an **Android** app, the Firebase configuration file is a **google-services.json** file. The development team should place this file into the `android/app` directory of the Flutter project. Finally, to enable Firebase functionalities in the Android app, the development team must add the `google-services plugin` to the `Gradle` files. The Flutter documentation details the lines of code to add[2].
- For an **iOS** app, the Firebase configuration file is a **GoogleService-Info.plist** file. Just like the Android application, the development team should add this file into the `Runner/Runner` directory of the Flutter Application.

To use Firebase for Flutter, it is necessary to add both the Android app and the iOS app to the Firebase project. Once these applications are installed, the company can install the Firebase packages it needs. The development team needs to insert the name of the packages required in the `pubspec.yaml` file of the Flutter project. For example, Red Nuclear Monkey's development team required the auth package of Firebase, and the team specified this package name in the `pubspec.yaml`:

Listing 5.1: Firebase Authentication package in pubspec.yaml

```
dependencies:
  flutter:
    sdk: flutter
  firebase_auth: ^0.16.0
```

[1]Firebase Console - `https://console.firebase.google.com/?hl=fr`

[2]Firebase documentation - `https://firebase.google.com/docs/flutter/setup?platform=android`

The Firebase documentation provides the packages available for Flutter[3]. Once the packages have been entered in the `pubspec.yaml` file, the development team must run the `flutter get packages` command to install these packages within the Flutter project and thus be able to use the features of Firebase.

5.2.2 Setting up Back4App/Parse

To use Parse functionalities through Back4App, the company must first create an application on Back4App. Next, the development team needs to add the Parse SDK[4] to its flutter project's `pubspec.yaml` file, then run the `flutter get packages` command to install this package. Red Nuclear Monkey use the version 1.0.26 of the Parse SDK. Finally, unlike Firebase, the development team must add lines of code to initialize Parse with the server being used, e.g. Back4App, and to configure Parse. The company must perform the initialization and configuration of Parse at the beginning of the application's execution. The example below illustrates the initialization and configuration of Parse for Alacrity project:

Listing 5.2: Initialization of Parse and Back4App

```
await Parse().initialize('keyParseApplicationId',
    'keyParseServerUrl',
    masterKey: 'keyParseMasterKey', // Required for Back4App
        and others
    debug: true,
    coreStore: await CoreStoreSharedPrefsImp.getInstance());
```

To initialize Parse SDK developers must provide the application ID and the URL of the server to use. With Back4App, developers can find the application ID in the Back4App project previously created and then in `Server Settings > Core Settings > Settings`. The server URL of Back4App is: `'https://parseapi.back4app.com/'`. As the Parse SDK indicates, the development team must fulfil the `masterKey` field to use Back4App services. Developers can find this key at the same place as the application ID. Finally, this value of the `coreStore` allows the use of the secure storage of Parse. Now, when running the application, Parse is initialized with a certain configuration allowing the development team to use the services provided by

[3]Firebase packages for Flutter - `https://firebaseopensource.com/projects/firebaseextended/flutterfire/`

[4]Parse SDK for Flutter Documentation - `https://pub.dev/packages/parse_server_sdk`

Back4App. The Parse SDK documentation provides more parameters for the configuration.

5.2.3 Authentication

As seen previously in Chapter 3, Alacrity requires the use of the Authentication feature. This functionality allows the application to know the identity of users to save their data in a cloud database and to grant users to use the application with their data on multiple devices.

To access Alacrity features, users must register. For user authentication, the application retrieves a user's credentials. These credentials can be the user's name, email and password or an Open Authentication (OAuth) token that comes from a federated identity provider (IdP) such as Facebook, Google or Apple. A token is a cryptographically signed document containing the claims of the client. Claims are the information about the identity that is calling the server. For authentication, Alacrity uses two methods that work differently:

- Basic authentication with an email and a password
- Authentication via an OAuth Provider

Figure 5.1 illustrates basic authentication:

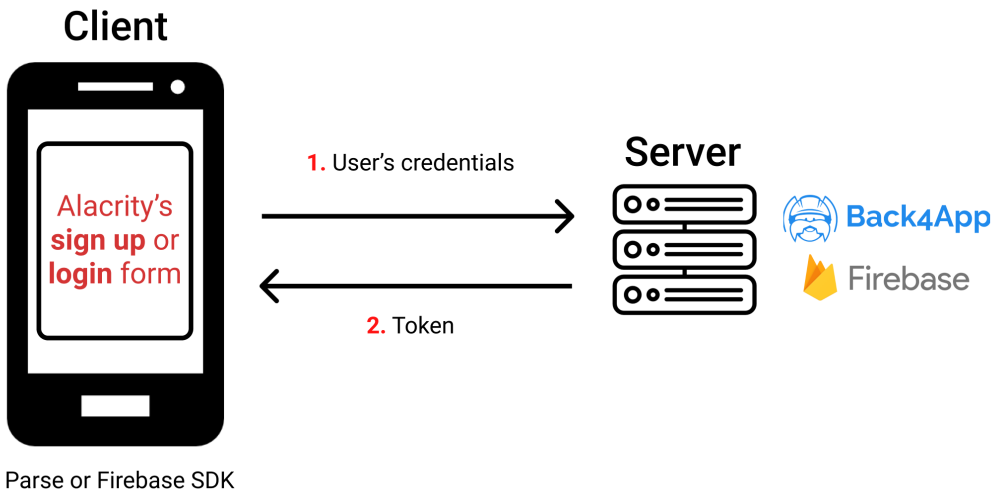


Figure 5.1: Basic authentication with email and password.

1. The client provides its credentials using a form. These credentials are sent securely to the server via the sign-up or login methods offered by the Firebase and Parse SDKs.

2. Then, the Firebase or Parse server validates its credentials and sends an authentication token back to the Flutter application.

Thus, after authenticating a user, the front-end of the application can access the user's data from this token. Finally, each time the user makes a request to the Parse or Firebase server via the frontend, for example by creating a top-goal, this token is used to prove the identity of the user and thus allow the modification or creation of the data for this specific user.

Figure 5.2 illustrates authentication via an OAuth Provider:

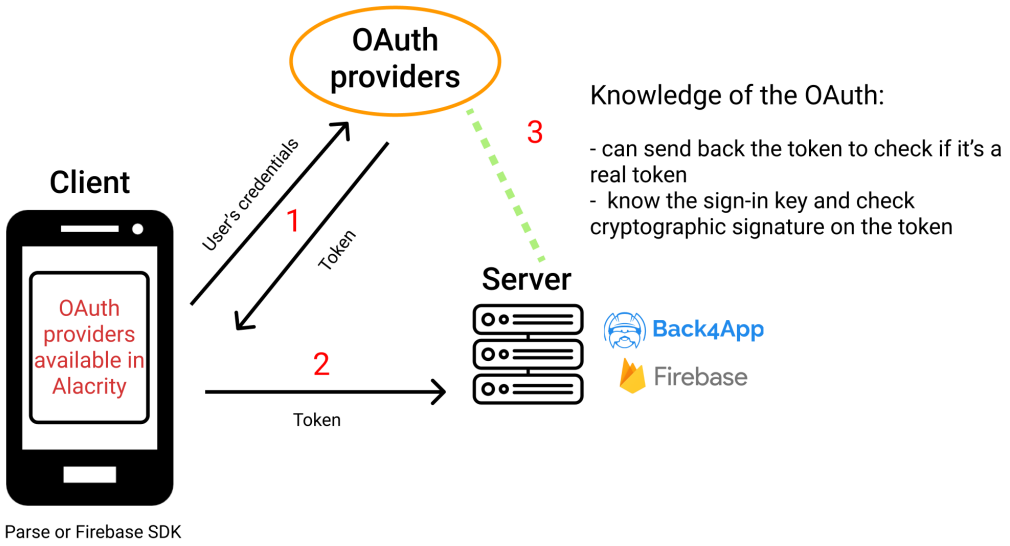


Figure 5.2: Authentication with OAuth provider.

1. The client receives the authenticate token directly from the identity provider, by providing to the IdP its credentials, e.g. its email and password.

2. Then, the client presents the token provided by the identity provider to the Firebase or Parse server.

3. The server can then validate the token since the server has a trust relationship with the identity provider. The server can send this token

to the identity provider to verify if it is correct or use the knowledge of the identity provider's sign-in key to verify the cryptographic signature on the token.

Once the server validates the token, the server uses this token as a proof of the user's identity.

The Alacrity development team created a single 'auth.dart' file that represents an 'AuthService' class containing all the methods and code necessary for authenticating a user. This class extends ChangeNotifier to notify listeners of changes related to the authentication of a user. Therefore, this file includes the multiple login methods presented in Chapter 3, the registration method and the sign out function. Finally, this file also includes the recovering password function for a user. All the functions implementing the authentication functionality are asynchronous methods. To perform asynchronous operations in Dart, the development team uses the `Future class` and the `async` and `await` keywords[5]. `Future` represents the results of asynchronous operations and has two states: completed or uncompleted. To define an asynchronous method, developers need to add `async` keyword before the function body. To get the result, developers need to add `await` keyword before getting the result. This chapter offers concrete examples of functions using `Future class`, `async` and `await` keywords.

5.2.3.1 Authenticate features with Firebase

For Firebase, the `AuthService class` contains three attributes:

- `FirebaseAuth _auth` which represents the instance of the Firebase Authentication service and which therefore allows the use of the methods provided by the Firebase service,
- `String pseudonym`, which illustrates the nickname of a user,
- `FirebaseUser fu`, which serves as the current Firebase user.

This class has a constructor that includes subscription to a stream provided by Firebase to listen to changes on the current Firebase user. The Firebase Auth Service emits an event every time a user logs in or logs out. So the Flutter application receive these events and can determine wheter a user is logged in or not. As soon as a user is connected or disconnected, the

[5]Asynchronous programming: futures, async, await - `https://dart.dev/codelabs/async-await`

value of the current Firebase user changes. Based on the value of the current Firebase user, the application renders the Authenticate screen or Goals screen. The example below offers the implementation of the AuthService constructor.

Listing 5.3: AuthService constructor with Firebase

```
class AuthService extends ChangeNotifier {
  AuthService() {
    _auth.onAuthStateChanged.listen((FirebaseUser _fu) {
      printer.w('Auth - New auth event!');
      fu = _fu;
      notifyListeners();
    });
  }
```

The AuthService class also contains:

- A function returning a boolean allowing to know if a user is authorized to access the Goals view. This function checks if the current Firebase user is not null and if the user has a UID and an email. If the value of Boolean is true, the Firebase user is logged in and has permission to access the Goals screen. Otherwise, the user is not logged in and is not authorized, so Alacrity renders the Authenticate screen.
- A function returning the UID of the current user and a function returning the email of the current user.
- A `loginWithEmailAndPassword` function which takes an email and a password as parameters. Then, this function uses the functionality of registering a Firebase user through the instance of the Firebase Authentication service by providing the email and password given as a parameter. The example below illustrates the use of the Future class and the async and await keywords to implement this function:

Listing 5.4: loginWithEmailAndPassword function

```
  Future<void> logInWithEmailAndPassword(String email,
     String password) async {
    try {
      await _auth.signInWithEmailAndPassword(email: email,
         password: password);
    } catch (e) {
      printer.e(e);
    }
  }
```

So if the log in via the Firebase function fails, the function prints the error.A `logInWithFacebook` function. *This feature requires the installation of the* ***`facebook_login`*** *package for Flutter*[6]*. Besides, to use the Facebook Login package, developers must register and configure their Flutter app with Facebook. This thesis does not present these steps. To use authentication with Facebook for Firebase, developers must activate the authentication method with Facebook via the Firebase console in the* ***`Authentication > Sign-in method`*** *menu. Firebase requires the app ID of the Facebook app and the secret app that represents the sign-in key of the OAuth Provider, e.g. Facebook, to decrypt the token.* This function has no parameters. This method uses the Facebook login, then stores the value of the Facebook login in a result variable. If the user is logged in with Facebook, the function retrieves the token to get the user's credentials. Finally, the function uses, thanks to the AuthService instance, the Firebase authentication method with credentials. After authentication with Firebase, the function retrieves the username by storing it in the pseudonym attribute of the AuthService class. The username is retrieved by making a request to the Facebook graph. The code below illustrates these steps:

Listing 5.5: Part of logInWithFacebook function

```
final FacebookLogin facebookLogin = FacebookLogin();
final FacebookLoginResult result =
await facebookLogin.logIn(<String>['email']);

if (result.status == FacebookLoginStatus.loggedIn) {
    final String token = result.accessToken.token;

    final AuthCredential credential =
        FacebookAuthProvider.getCredential(accessToken:
            token);

    await FirebaseAuth.instance.signInWithCredential(
       credential);

    final http.Response graphResponse = await http.get(
        'https://graph.facebook.com/v2.12/me?fields=name,
            first_name,last_name,email&access_token=$token')
            ;
    final dynamic profile = jsonDecode(graphResponse.body);
    pseudonym = profile['name'].toString();
```

[6]Facebook Login Package - `https://pub.dev/packages/flutter_facebook_login`

```
}
```

If the user did not connect during the Facebook login, i.e. he cancelled or he entered wrong credentials, the function prints an error.A `logInWithApple` function. *This method requires the installation of the sign_in_with_apple package. Also, this feature requires some configurations to work. The package documentation details these configurations*[7]*. Alacrity does not use this feature for Android and iOS versions below version 13.* This function is very similar to that of Facebook. The method uses the function provided by the package to login to Apple. If the user logs in with Apple, the function retrieves the credentials and uses the Firebase authentication method with those credentials. This method also retrieves the username in the pseudonym attribute using Apple credentials. If the user did not connect during Apple's login, i.e. he cancelled or filled in wrong credentials, the function prints an error. A `registerEmailPassword` function, allowing to register a new user using an email and a password. This function takes a user's name, email and password as parameters and then calls the user creation method provided by the Firebase Auth Service. The method provided by Firebase Auth Service requires email and password. Finally, the pseudonym attribute stores the name of the user. The example below shows the implementation of this function.

Listing 5.6: registerEmailPassword function

```
Future<void> registerEmailPassword(
      String email, String password, String name) async {
    try {
      await _auth.createUserWithEmailAndPassword(
          email: email, password: password);
      pseudonym = name;
    } catch (e) {
      printer.e(e);
    }
  }
```

– A `signOut` function, allowing to disconnect the current user. This function does not take any parameters. This method only calls the signOut function provided by Firebase Auth Service.

[7]Sign In With Apple Package - `https://pub.dev/packages/sign_in_with_apple`

5.2.3.2 Authenticate features with Back4App/Parse

For Back4App, the `AuthService class` also contains three attributes:

- `ParseUser parseUser` which represents a Parse user,
- `String pseudonym`, which represents the nickname of a user,
- `String auth`, which represents a user's means of authentication.

Alacrity needs the auth string attribute for Back4App, as this service works differently compared to Firebase. Indeed, with Firebase, a user can register with his email and then use the Facebook or Apple authentication means with this same email and his data will be merged. However, with Back4App, a user cannot register with his email and then use another means of authentication. Therefore, this string makes it possible to record the means of authentication of a user. Therefore, each time a user registers for the first time, the AuthService class directly registers the user in the database and also registers the means of authentication of that user. So when a user logs in, the authentication functions verify that the user exists and verify that the user has registered with the authentication method he is using. If not, an error message is sent to the user, telling him that he registered for the first time with another authentication method.

The `AuthService class` has a constructor that does not provide a stream, but a function. Parse does not produce a stream functionality like Firebase, so the development team found a way to provide the same work as a stream. Hence, the constructor has a `triggerUpdate function` which checks that the current user of Parse is not null and has a UID. If the user is not null, the parseUser attribute of the AuthService class takes the value of the current user of Parse and notifies its listeners. The sample code below shows the implementation of this constructor:

Listing 5.7: AuthService constructor with Parse

```
class AuthService extends ChangeNotifier {
  Future<void> triggerUpdate() async {
    final ParseUser user = await ParseUser.currentUser() as
        ParseUser;
    if (user != null && user.objectId != null) {
      parseUser = user;
      notifyListeners();
    }
  }
```

Unlike the authentication service provided by Firebase, Parse uses authentication methods directly on ParseUser objects. A ParseUser is built with three mandatory attributes: an email, a unique identifier, and a password. These attributes are strings. To have a unique identifier, the Red Nuclear Monkey development team used the email of a user.

The AuthService class also has the following functions:

- A function returning a `boolean` allowing to know if a user is authorized to access the Goals view. This function checks if the current `ParseUser` is not `null` and if the user has a `UID` and an `email`. If the value of boolean is `true`, the Parse user is logged in and has permission to access the Goals screen. Otherwise, the user is not logged in and is not authorized, so Alacrity renders the Authenticate screen.

- A function returning the `UID` of the current user and a function returning the `email` of the current user.

- A `loginWithEmailAndPassword` function that takes `ParseUser` as a parameter. This `ParseUser` is initialized in the `log_in.dart` file when the user fills in the form fields. Then, this function uses the function of registering a user on the `ParseUser` given as a parameter. If the authentication is successful, the function calls the `triggerUpdate` method to notify listeners that a user is connected. Otherwise, the function returns an error message.

- A `logInWithFacebook` function. This function works the same as that implemented using Firebase. It therefore also requires linking the Android and iOS application with Facebook and installing the `facebook_login` package. To activate the login with Facebook, developers must enter the application ID of their Facebook App in the server settings menu of the application on Back4App console.

  Unlike the function implemented with Firebase, when the user successfully connects with Facebook, the function implemented with Parse checks if the user has already registered by performing a query in the database.

  - If the user is not registered, the function creates the user in the database by calling the `fetchParseUserData` function. This section presents the `fetchParseUserData` function below. The function set the `string auth` as 'Facebook'. Finally, the function calls `triggerUpdate` to notify listeners that a user is connected.

  - If the user is registered, the function checks that the means of authentication is 'Facebook'. If the means of authentication is not equal to 'Facebook', the user does not log in and receives a message telling him to choose another means of authentication. Otherwise, the user logs in successfully, and the function calls `triggerUpdate` to notify listeners that a user is logged in.

- A `registerEmailPassword` function, allowing to register a new user using an email and password. This function takes as a parameter the `name`, and the `ParseUser` created beforehand when the user has filled in the registration form. The function performs the `signUp` method provided by Parse SDK on the `ParseUser`. If the `signUp` is successful, the function calls the `fetchParseUserData` method to register the user in the database by providing the name, the `ParseUser` and the means of authentication, e.g. 'Normal' in this case, then calls the `triggerUpdate` method to notify that a user is now logged in. Else, the function returns an error.

- A `signOut` function, which takes no parameters and performs on the current `ParseUser` the `signOut` method provided by Parse SDK. The function then calls `triggerUpdate` to notify that no user is logged in.

- A `fetchParseUserData` function which takes as parameters a `ParseUser` and the attributes of a user, such as for example his means of authentication. This function stores the attributes of a `ParseUser` in the Parse User database. The example below shows the implementation of this function:

Listing 5.8: fetchParseUserData function

```
  Future<void> fetchParseUserData(
      ParseUser parseUser, String name, String auth,
      {String email}) async {
    parseUser.set('name', name);
    parseUser.set('auth', auth);
    if (email != null) {
      parseUser.set('email', email);
    }
    await parseUser.save();
  }
}
```

Back4App provides the possibility of being able to connect using Apple as a provider. However, the Parse SDK for Flutter does not provide this feature.

Consequently, the version of Alacrity that uses Parse and Back4App do not provide a login with Apple.

5.2.4 Password reset

This feature allows Alacrity users who have forgotten their password to receive an email so they can set a new password. The Red Nuclear Monkey development team defined this feature in the `auth_dart` file since this feature is related to user authentication. Also, this function does not check that a user's email address exists in the database.

5.2.4.1 Reset password feature with Firebase

The function to reset a user's password invokes the password reset function provided by Firebase Auth Service taking a user's `email` as a parameter. The feature provided by Firebase only requires a user's `email` as a parameter.

In the Firebase console and in the Authentication section, it is possible for developers to change the email template that users will receive when asking to reset their passwords. Figure 5.3 illustrates this possibility and presents the default message provided by Firebase.

5.2.4.2 Reset password feature with Back4App/Parse

The reset password function for Parse directly calls the function provided by Parse SDK. In contrast to Firebase, the function provided by the Parse SDK required a `ParseUser`. As defined before, a `ParseUser` is created by providing a `name`, a `password` and an `email`. Since the user who calls that function has forgotten his password, it is therefore impossible to create the corresponding `ParseUser`. Developers must create a `ParseUser` with `null` attributes for the `password` and the `name` and set the `email` attribute as the email of the user that has forgotten his password. This practice did not appear to be common for the Alacrity development team.

Back4App allows developers to change the email template regarding password restoration like Firebase. Figure 5.4 shows the default message provided by Back4App.

5.2.5 Database

This section presents the interaction of the Alacrity application with the Firebase and Parse databases. The models presented in Chapter 3 interact

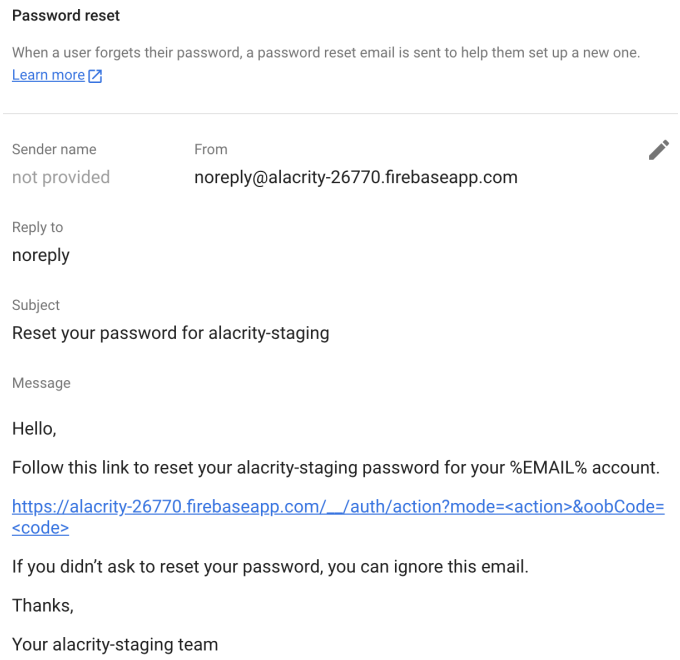


Figure 5.3: Firebase Reset Password email template.

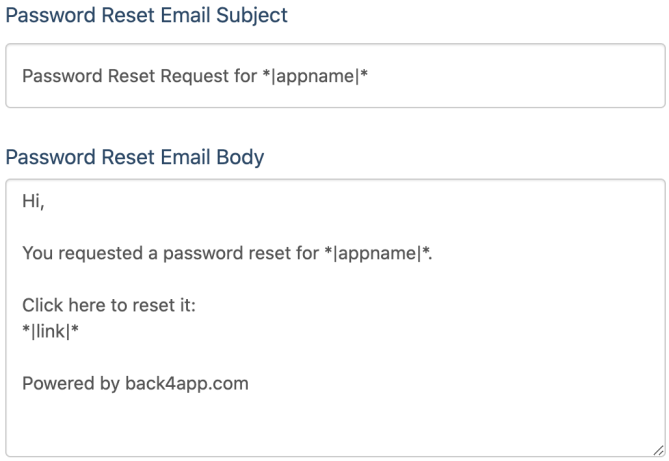


Figure 5.4: Back4App Reset Password email template.

with the BaaS databases, by creating, updating and deleting data. Besides, managers have a direct link with the databases by listening to the change of data. This thesis only presents the implementation of certain features of the TopGoal model and the TopGoal manager illustrating the interactions with

the database previously defined in Chapter 3. This research does not present the other models and managers since their implementations are very similar.

5.2.5.1 Firebase Database: Firestore

The Alacrity development team used the Firebase real-time database. This database is a NoSQL database allowing a rapid execution of operations [40]. The real-time database stores data in JSON objects. JSON is qualified as easy to read and understand [40]. This database provided by Firebase allows developers to store and synchronize user data in real-time. This feature allows users to access their data on any device. When data is updated in the database, all users interested in this data change are notified on their devices within a few milliseconds [24]. Also, the database SDK uses a local cache on the device when users are offline to serve and store changes. This functionality allows synchronizing automatically the local data of the users when they come back online [24].

5.2.5.2 Back4App/Parse Database

Back4App does not provide precise information on its database. An article on the Back4App blog describes only that Parse Server uses MongoDB to store data and Amazon S3 bucket to store files[8]. Additionally, in their documentation, Back4App states that developers can store any data that can be JSON-encoded, i.e. objects such as strings, numbers, booleans and dates[9].

5.2.5.3 Database interaction for TopGoal Model

The implementation of the `TopGoal` model is similar for the two Backend-as-a-Service. Indeed, the `TopGoal class` contains for Parse and Firebase the same functions and attributes. The changes concern the calls for the database functionalities provided by the two services Parse and Firebase. The `TopGoal class` contains:

- The attributes of a TopGoal, such as the name of a TopGoal represented by a `string`, its creation date represented by a `Date` and a list of motivation and to-Donations represented by `lists of strings`. For Firebase, the class also contains a `CollectionReference topGoalCollection` as an attribute. This attribute can be used for adding documents,

[8]Parse Vs. Firebase - Back4App - `https://blog.back4app.com/parse-vs-firebase/`

[9]Back4App documentation - `https://www.back4app.com/docs/get-started/add-get-data-with-back4app`

getting document references, and querying for documents. Here, this `CollectionReference` is a reference from the 'TopGoals' collection.

- A constructor, allowing to create a `TopGoal` with the attributes defining it.
- A `create()` function, which takes no parameters and returns a `boolean`. This function takes the current date and then tries to create a `TopGoal` with its attributes in the BaaS database:
  - For Firebase, the function creates a `DocumentReference` which refers to a document location in a Firestore database and which can be used to create, read or listen to a location. This `DocumentReference` keeps as value the operation performed on the `CollectionReference topGoalCollection`. The operation performed on this `CollectionReference` is `add()`, provided by Firebase SDK. With this operation, we add to the 'TopGoals' collection in Firestore a new `TopGoal` document with its attributes. The example below shows an implementation of saving a document to the Firebase database.

Listing 5.9: Create TopGoal in FireStore

```
final DocumentReference doc =
          await topGoalCollection.add(<String, dynamic>{
        'value': value,
        'motivations': motivations,
        'toDonts': toDonts,
        'startDate': now,
        'createdBy': createdBy,
      });
```

  - For Parse, the function creates a `ParseObject('TopGoal')`. This is the way to save a `TopGoal` object in Back4App database. Then, the function sets the attributes of the `TopGoal` to this object and then saves this object in the database by calling the `save()` function provided by Parse SDK on this ParseObject. The example below illustrates the implementation of saving an object in the Back4App database.

Listing 5.10: Create TopGoal in Back4App database

```
final ParseObject newTopGoalParse = ParseObject('
    TopGoal')
```

```
    ..set('value', value)
    ..set('motivations', motivations)
    ..set('toDonts', toDonts)
    ..set('startDate', now)
    ..set('createdBy', createdBy)
    ..set('createdAt', now);
  await newTopGoalParse.save();
```

If the operation is successful, the function returns `true`, otherwise, the function returns `false`.

- An `update()` function, which takes as parameters a `list of motivations string` and a `list of to-Dont's string`. As a reminder, a user can only change the motivations and to-Dont's of a `TopGoal`, as shown in figure 3.12. This function returns a `boolean`.

  - For Firebase, the function selects the `TopGoal document` to modify on the `TopGoalCollection CollectionReference` using its `ID` and performs the `setData()` method which allows changing the value of the attributes of this document.
  - For Parse, the function creates a `ParseObject ('TopGoal')` having the `ID` of the `TopGoal` to modify and sets the new attributes to this object. Finally, the function saves the changes in the Back4App database by calling the `save()` method on this ParseObject.

  If the modification of the `TopGoal` is successful, the function returns `true`, otherwise `false`.

- Finally, the `TopGoal class` contains the `delete()` function allowing to delete a `TopGoal` from the BaaS database. This function does not take any parameters and returns a `boolean`.

  - Like the `update()` function, for Firebase, the function selects the `TopGoal document` to delete on the `topGoalCollection CollectionReference` using its `ID` and performs the `delete()` method provided by Firebase SDK, allowing the suppression of a document in the collection.
  - For Parse, the function creates a `ParseObject('TopGoal')` with the `ID` of the TopGoal to delete and then performs the `delete()` operation provided by Parse SDK.

  If the removal of the `TopGoal` is successful, the function returns `true`, otherwise `false`.

5.2.5.4 Database interaction for TopGoalManager

This section presents the interaction of the `TopGoalManager class` with the BaaS database and some functions used by the Goals screen. The implementation of the `TopGoalManager class` differs slightly between the two Backend-as-a-Service. As a reminder, the `TopGoalManager class` extends `ChangeNotifier` and allows screens Goals to have access to this class via a Provider. Thus, the Goal screen can create an instance of this class to have access to the attributes of the `TopGoalManager class` and to its functions. The `TopGoalManager class` provides a list of a user's TopGoals in real-time. Thus, for each event linked to a user's TopGoal list, i.e. the creation or deletion of a `TopGoal`, the Goal view adapts in real-time.

For **Firebase**, the `TopGoalManager class` contains as attributes:

- A `string` representing the ID of a user,
- A `TopGoal list`, representing a user's list of TopGoals,
- A `stream`, allowing you to listen in real-time to changes to a user's TopGoals collection,
- A `CollectionReference`, which is a reference to the 'TopGoals' collection in Firestore.

With these attributes, the constructor of the `TopGoalManager` assigns the `stream` to listen to an event on a user's TopGoal collection. To do so, the function uses the `CollectionReference` and performs a query to listen only to TopGoals that are created by the user, i.e. which contain the user ID for the 'createdBy' field. Thus, each time the TopGoals collection is modified, i.e. creation, deletion or update of a TopGoal, an `onStreamUpdate()` function is triggered.

The `onStreamUpdate()` function retrieves the most recent user's TopGoal collection. So each time a user deletes or adds a TopGoal, the collection attribute of the TopGoalManager class contains the new list of TopGoals of a user and notifies listeners. By notifying listeners, Flutter will rebuild the widgets that listen for this change and will allow the user to see changes to their TopGoal list in real-time in the Goals view.

For **Parse**, the implementation is different. As explained earlier in this Chapter, Parse does not provide streams like Firebase. However, Parse provides Live Queries. Live Queries allow developers to subscribe to a `Parse.Query` that they are interested in. Once subscribed, the server will

notify clients whenever a `Parse.Object` that matches the `Parse.Query` is created or updated, in real-time[10]. To activate Live Queries, developers must add a `liveQueryUrl` argument to the configuration of Parse, which is provided by the web hosting service in the application configurations of the web hosting service. Besides, for Back4App, it is necessary to navigate in the application options via the Back4App console, to activate the Live Queries functionality on the collections that require this service. For example, the Alacrity development team enabled the Live Queries functionality for the 'TopGoals' collection. The `TopGoalManager class` contains these attributes for Parse:

- A `string` representing the ID of a user,
- A `TopGoal list`, representing a user's list of TopGoals,
- A `<ParseObject> Subscription` which allows you to subscribe to a Parse object,
- A `LiveQuery` defining a query which will be listened to continuously.

The `TopGoalManager class` constructor for Parse contains two functions: `listenTopGoalCollectionUpdate()` and `getTopGoalCollection()`.

The `listenTopGoalCollectionUpdate()` function initializes a `LiveQuery` on a user's TopGoals, and perform the same work as the stream for Firebase. Therefore, The development team had to subscribe to update, create and delete events of a user's TopGoal to properly change a user's TopGoal list in real-time. Each time an event is detected, the `getTopGoalCollection()` function is triggered. The example below illustrates the implementation of this function.

Listing 5.11: listenTopGoalCollectionUpdate()

```
Future<void> listenTopGoalCollectionUpdate() async {
    liveQuery = LiveQuery();
    final QueryBuilder<ParseObject> query =
        QueryBuilder<ParseObject>(ParseObject('TopGoal'))
          ..whereEqualTo('createdBy', userId);
    subscription = await liveQuery.client.subscribe(query);

    subscription.on(LiveQueryEvent.update, (ParseObject
       goal) {
      print('*** UPDATE ***: ${DateTime.now().toString()}\n
          $goal ');
```

[10]Live Queries Parse - https://docs.parseplatform.org/parse-server/guide/#live-queries

```
      printer.w('UPDATE TOPGOAL MANAGER');
      getTopGoalCollection();
    });
    subscription.on(LiveQueryEvent.create, (ParseObject
        goal) {
      print('*** CREATE ***: ${DateTime.now().toString()}\n
          $goal ');
      getTopGoalCollection();
    });
    subscription.on(LiveQueryEvent.delete, (ParseObject
        goal) {
      print('*** DELETE ***: ${DateTime.now().toString()}\n
          $goal ');
      getTopGoalCollection();
    });
  }
```

The `getTopGoalCollection()` function queries a user's TopGoals. If the query finds TopGoals, the function adds those TopGoals to the user's list of TopGoals. Like Firebase, each time a user makes a change regarding a TopGoal, Flutter will rebuild the Goal view to allow the user to get the latest changes.

The implementation of the TopGoalManger class is the same for the rest of the functions between Firebase and Parse. The class provides four other functions that are used on the Goals screens:

- A `getTopGoalById` function which returns a `TopGoal` and which takes a `string id` as parameter. This function returns the `TopGoal` with the `id` given as a parameter in a user's TopGoal collection.
- A `createNewTopGoal` function, which takes as parameters the attributes needed to create a `TopGoal`. This function creates the `TopGoal` object with the attributes given in parameter and performs on this object the `create()` function provided by the `TopGoal class`, which represents the model of a TopGoal, and which allows creating the TopGoal in the database of BaaS.
- An `updateTopGoal` function, which takes the `TopGoal id`, the new `motivation list` and the new `to-Dont's list` as parameters. The function retrieves the TopGoal using the `getTopGoalById` function by giving it `the TopGoal id` as a parameter. Then, the function performs the `update()` operation of the `TopGoal` class by providing it with the list of motivation and to-Dont's.

- A `deleteTopGoal` function, which takes the `id` of the TopGoal as a parameter. The function retrieves the TopGoal using the `getTopGoalById` function and then performs the `delete()` operation of the `TopGoal class`.

The screen Goals calls the createNewTopGoal, updateTopGoal, and deleteTopGoal functions through the TopGoalManager class instance. Thus, when a user clicks on an operating button, i.e. creating, editing or deleting a TopGoal, the Goal view performs the corresponding functions.

5.2.6 Notifications

Notifications are messages that a mobile user receives on the locked screen of its smartphone or via pop-up if the phone is active. Pop-ups usually have an alert form and are at the top of the phone screen. Users get these notifications when new data is available for an application, even if this app is not running in the foreground. Two categories represent these alerts: Push notifications and Local notifications.

- Push notifications, also known as Remote notifications, are messages that come from a remote server and sent directly to users' phones in actual time. These messages inform an event that is happening or has just happened, such as for example the reception of a Short Message Service (SMS), a missed call or a package that has just arrived at a relay point. They require an infrastructure, i.e. a server to contain push notifications and certain triggers to send these notifications. Therefore, push notifications expect an internet connection.

- Meanwhile, local notifications come directly from a mobile application, i.e. they do not come from a remote server but from the phone itself. Developers plan these notifications in several ways during app development. For example, when a user has not used the application for a certain period, a local notification can notify him informing how many hours he has not launched the app. Unlike remotes notifications, local notifications do not require an internet connection.

These two types of notifications require adding code in the development of an application, and it is critical to know when to use the right variety of notification. Application developers cannot control and predict when a user will use the application.

The data necessary to schedule local notifications are accessible only when the user opens the application. For example, retrieve the time when the user

closes the application to warn him if he does not reopen it for 24 hours from this time. The system to unlock rewards in Clash Royal, a mobile game developed by Supercell, illustrates a concrete case of local notification. When a player unlocks a chest, the game set a timer and at the end of it the user receives a notification to collect his reward.

Regarding push notifications, a notification appears to the user when new data is available for him. The user does not need to check for new data himself. For example, when someone receives an email, a notification notifies this person informing that there is a new email. This person does not need to refresh its emails every minute.

5.2.6.1 Local notifications

As explained previously, local notifications do not require the use of Backend-as-a-Service features. Thus, this thesis does not present the implementation of local notifications implemented for the Alacrity application. Red Nuclear Monkey Company used the `flutter_local_notifications package`[11] to set up these local notifications. Each day, a user receives a notification around 12:00 a.m. (UTC +3) reminding him how many days he has left before the end of his TopGoal. Besides, each time a user starts a session, a notification notifies that user when the current session is over.

5.2.6.2 Push notifications

The Red Nuclear Monkey company requires push notifications to notify its users of new features or updates to the application. Push notifications become more and more popular [9] and can be an effective way to drive user engagement [29]. However, they can have the opposite effect if companies use these push notifications incorrectly, which leads to user dissatisfaction and a uninstallation of the application [29]. These push notifications are a service provided by Backend-as-a-Service as explained in Chapter 4.1. This thesis shows the push notification services offered by Firebase and Back4App using them for the Alacrity application.

To use the **Firebase Push Notifications feature**, developers must install the `firebase_messaging package` and then configure this SDK by following the official Firebase documentation for this package. This package requires you to configure specific files for Android and iOS to function properly.

[11]Flutter Local Notification package - `https://pub.dev/packages/flutter_local_notifications#-readme-tab-`

Firebase Cloud Message is a smart way to send push notifications to users of an application registered on Firebase. This Firebase service allows sending push notifications to a single device or user, to a group of users who have a common topic, or to a user segment based on analytical data previously collected by Firebase. In the Firebase console, developers can perform a push notification to target a user segment by selecting several criteria such as language and location of users. Based on application analytics, Firebase allows developers to see the number of people who will receive the notification based on the criteria selected and the percentage that this segment represents out of all users who use the application. Figure 5.5 illustrates a push notification via the Firebase Console which is addressed to a user segment using the application on iOS with the French language and located in Finland. Further, with the Firebase Predictions tool, developers can dynamically segment their users and send notifications based on the engagement a company is trying to achieve.

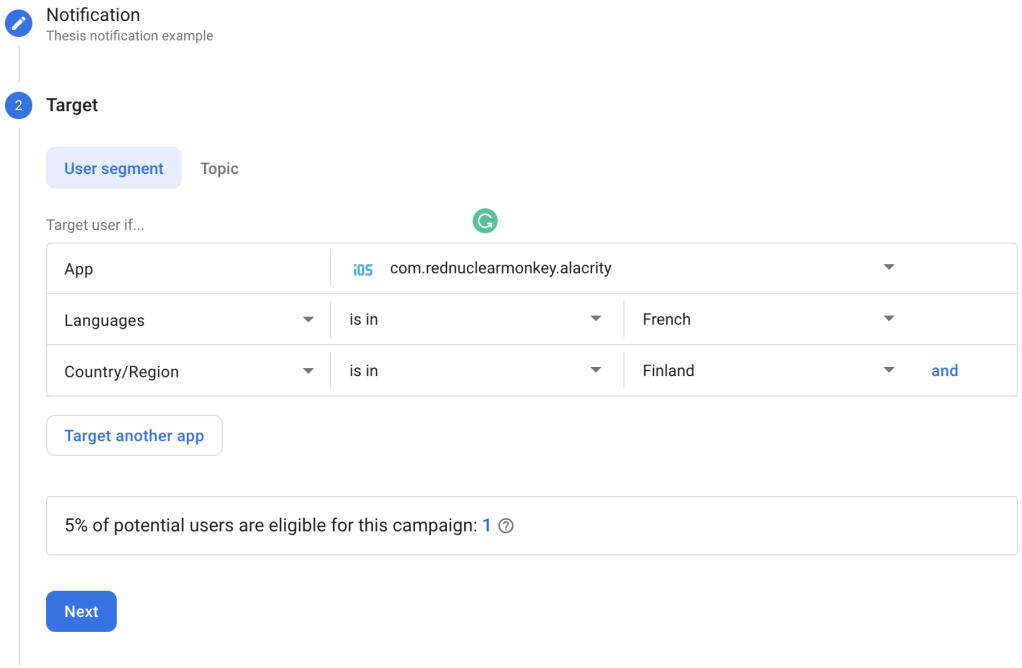


Figure 5.5: Firebase Console Push Notification for iOS with a user segment.

Once the developers installed and configured the package, they can test and send push notifications to devices connected to the app in three different ways:

- via the Firebase console (UI)

- via Firebase Cloud Functions: In most applications, companies send push notifications by using cloud functions, and more particularly when companies target specific devices or users.
- via a terminal: The following command illustrates sending the same push notification using the Firebase Condole shown in Figure 5.6.

Listing 5.12: Firebase push notification via terminal

```
DATA='{"notification": {"body": "Thesis example Android","
    title": "Thesis example Android"}, "priority": "high", "
    data": {"click_action": "FLUTTER_NOTIFICATION_CLICK", "
    id": "1", "status": "done"}, "to": "<FCM TOKEN>"}'
curl https://fcm.googleapis.com/fcm/send -H "Content-Type:
    application/json" -X POST -d "$DATA" -H "Authorization:
    key=<FCM SERVER KEY>"
```

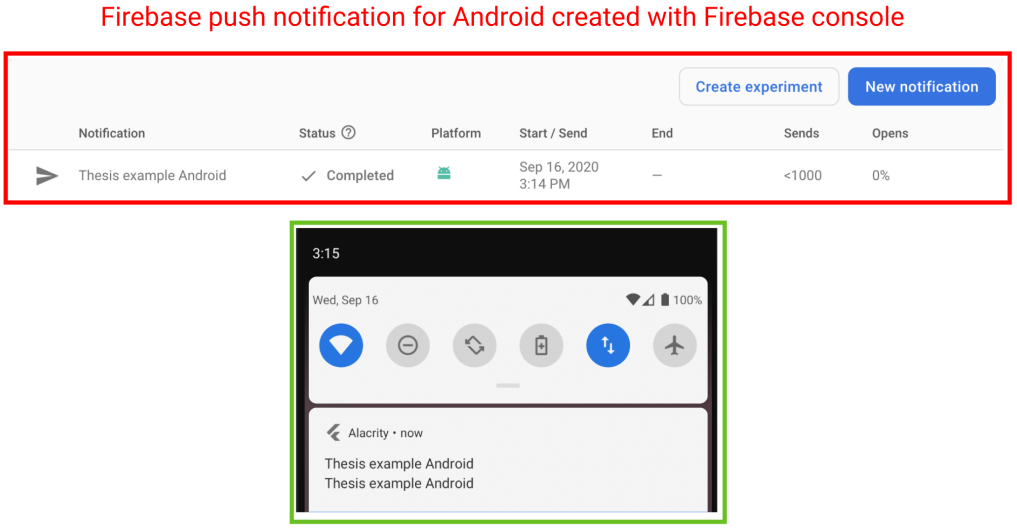


Figure 5.6: Firebase Console Push Notification for Android.

Regarding the **push notification functionality of Back4App**, developers required to use the services of Firebase. The Back4App platform does not provide a push notifications service by itself. Thus, to access push notifications by following the Back4App documentation[12], developers must

[12]Back4App Push Notifications Documentation - `https://www.back4app.com/docs/android/push-notifications/parse-server-push-notifications`

register the Flutter application developed with Parse on Firebase. Developers should, therefore, proceed with the Firebase setup described earlier in this chapter. Also, this means that the development team integrates a new Backend-as-a-Service solution into their project, and now has the functionalities of Parse and the push notifications functionality of Firebase. To send push notifications to its users, developers must proceed in the same way as with Firebase, i.e. install the `firebase_messaging package` and use the different solutions of sending push notifications offered by Firebase.

The next chapter presents the most suitable Backend-as-a-Service solution for the Red Nuclear Monkey company, based on the result of using Firebase and Back4App through the multiple features offered in this chapter for the development of Alacrity.

Chapter 6

Results

This chapter gives the most suitable Backend-as-a-Service solution for Alacrity development based on the Red Nuclear Monkey criteria. This thesis presents the results in the use and implementation of the functionalities needed and cited in Chapter 5. Thus, this work presents the most relevant BaaS solution concerning setup, user authentication, interaction with the database, push notifications service and the price. This section also offers some additional data such as the number of lines of code and the final size of the application to mark the difference between the two BaaS and to point the impact that these two BaaS have on a final project.

Initially, the setups of the two Backend-as-a-Services were effortless to integrate into the Flutter project even though they had two different approaches. When creating the application on Back4App, Parse does not require including new files in the project, and only requires the installation of the Parse SDK package for Flutter compared to Firebase. Also, Firebase requires developers to initialize two applications in Firebase Console for iOS and Android, which is not the case for Back4App.

Regarding the authentication feature, one of the two Backend-as-a-Service solutions stands out. For basic authentication, i.e. that does not use providers, the implementation of functions is noticeably similar for both BaaS. The only difference is that Parse SDK performs the authentication functions on an initialized ParseUser with a unique username, password and email. This restriction made the implementation less familiar to the development team, especially for implementing the forgot password feature. Creating a null user to call the Parse feature leaves the development team perplexed. However, for the implementation of authentication functions using Providers, e.g. Facebook and Apple providers, Firebase stands out and is the most convenient solution in this functionality for Red Nuclear Monkey. Indeed, for the

authentication of Facebook with Back4app and Parse, the team development had to make several checks on the user to find out if he exists and if he has already logged in with this means of authentication. These checks are unnecessary with Firebase because Firebase links accounts that have the same email address. Besides, although Back4App provides authentication functionality with Apple, Parse's SDK for Flutter does not currently allow authentication with Apple. This lack is a real problem for the company Red Nuclear Monkey since it intends to put its application on the App Store and Google Play Store. Now, with Apple's policy, if an application provides several means of authentication with providers, this application must also provide authentication with Apple Sign In to be eligible on the App Store[1]. Consequently, if the Red Nuclear Monkey company uses Parse as BaaS, the company cannot upload its application to the Apple Store, and the company would face new challenges to counter this lack.

The database functionalities of the two Backend-as-a-Services allow Alacrity to get and update user data in real-time by offering two different approaches: streams for Firebase, and live queries for Parse. Parse makes it easy to manipulate data in users since Parse allows developers to manipulate objects such as strings, arrays or dates. In contrast, to achieve the same functionality as Firebase for real-time data, Parse uses live queries on the collections in the database. For example, for the TopGoal collection, Parse sets up a live query to listen to each event linked to a user's TopGoal which is characterized by three subscriptions to the live query and, therefore, three requests on the Back4App server. On the other hand, Firebase uses stream functionality. The developers only perform a request on the Firebase server to access to real-time data on a collection. Thus, the more the number of users increases, the greater the request gap between the two BaaS servers. Also, in Alacrity, developers use real-time data for the Session and User collections, which increases the number of requests for Parse compared to Firebase. Finally, Back4App allows the use of live queries only when developers activated this feature on the collections via the Back4App UI which makes this step is easily overlooked. The Parse SDK debugging did not warn of this oversight, which resulted in a slight loss of time in development. The different approaches used by Parse and Firebase make Firebase a better choice for the Alacrity application.

Regarding the price, the thesis presented in Chapter 4 the different offers of the two Backend-as-a-Service. For Red Nuclear Monkey, Firebase is the more beneficial of the two, since it charges users only for the resources used,

[1]Guideline for Sign In with Apple - `https://developer.apple.com/app-store/review/guidelines/#sign-in-with-apple`

whereas Back4App charges users monthly, and might charge them more depending on the offering they choose and resources used. Also, the number of requests is higher for Parse, which results in more resources used on the Back4App server. For a company with limited means, the option of Firebase appears to be the most attractive.

In total, to implement the authentication features and the User and TopGoal models and the TopGoalManager, the development team wrote 479 lines of code using Parse as the BaaS solution and 449 lines of code using Firebase. Parse requires slightly more code to implement the functionalities, but these lines of code are notably related to the verification of the registration of a user in the database and the verification of the means of authentication of a user in the implementation of the authentication functions. The difference is minimal and reflects that neither of the two Backend-as-a-Service takes longer than the other to implement the functionality required for Alacrity's development.

Finally, the final build size of the app, i.e. the production version, with Firebase is slightly larger than the app with Parse: 20.8 Mb for Android file (.apk) and 27.4 Mb for iOS (.ipa) with Firebase compared to 20.7 Mb for Android and 26.9 Mb for iOS with Parse. The team development measured the sizes of apps for Android and iOS following the official Flutter documentation[2]. Even if the application developed with Parse uses the push notification service and therefore the Firebase package for push notifications and Firebase setup, the Parse application is lighter than all the Firebase one. However, the size of the final applications is not a deciding factor as the difference is not significant and remains remarkably slight.

Table 6.1 summarizes the previous categories to determine which of the two services is more beneficial for Red Nuclear Monkey and illustrates the most suitable BaaS solution for each of its points. This table represents the most suitable BaaS by indicating the character 'X' for each feature.

To conclude, the work done during this thesis helps define Firebase as the most suitable BaaS solution for the Red Nuclear Monkey company. From a financial point of view, Firebase appears to be the most relevant solution and allows to use the fewest resources to get real-time data. Besides, Firebase is the only solution, regarding the two solutions studied, to offer authentication with Apple for its users. Thanks to this functionality, Firebase allows the Red Nuclear Monkey company to authenticate its users with other providers

[2]Measuring your app's size - Flutter Documentation - `https://flutter.dev/docs/perf/app-size`

and to put their application on the App Store. Using Parse for Flutter does not meet this company criteria.

Feature	Firebase	Parse - Back4App	Summary
Setup		**X**	Back4App and Parse require fewer steps to set up a Flutter application.
Authentication	**X**		Firebase is more complete and provides Apple ID authentication. Thanks to that, the application can be released in the App Store.
Password reset	**X**	**X**	Both BaaS are equivalent, even if the Parse implementation seems not familiar.
Database	**X**		Firebase requires fewer requests to get real-time data.
Push notifications	**X**		Back4App uses Firebase to provide this feature. Firebase cloud messages are complex and allow developers to target anyone.
Price	**X**		Firebase provides a more efficient offer for Red Nuclear Monkey as the platform charges only for the resources used.
Lines of code	**X**	**X**	No major difference in terms of lines of code.
Size of application	**X**	**X**	No major difference regarding the size of the final build.

Figure 6.1: Summary of the comparison.

Chapter 7

Conclusion

This thesis presented the implementation of two Backend-as-a-Service solutions for the Alacrity application of the Red Nuclear Monkey company. Red Nuclear Monkey is a small company with limited resources, which has been highly affected by the COVID-19 situation. Despite this health crisis, the company could develop an internal project. This paper has illustrated the strengths and weaknesses of the Flutter framework, which is a framework for developing hybrid applications. From the research done in Chapter 2, the thesis showed that Flutter is becoming a tough competitor for React Native and that in the years to come the framework could become the leader for Hybrid application development.

This work also offered a solution found by Red Nuclear Monkey to develop a robust Flutter application and thus be able to integrate with ease several Backend-as-a-Service solutions. With this solution, the company can easily switch between BaaS solutions without affecting the development of features that do not require the need for BaaS.

Besides, a review of narrative literacy from academic publications and industry literature has enabled this thesis to understand the motivations of companies in their choice of whether to use a Backend-as-a-Service solution. Even if Backend-as-a-Service solutions have many advantages like saving development time and cost, these solutions also have disadvantages that can hamper companies in their use. However, despite the cons, nowadays the cloud is omnipresent and large companies such as Spotify, Ubisoft, Snapchat, Netflix or Twitch trust and use its solutions (Google Cloud Platform and Amazon Web Services). Many solutions exist and are more or less robust, so as an application grows, companies can migrate to other BaaS solutions to get a more robust environment.

Finally, this thesis offers the company Red Nuclear Monkey the best BaaS solution between Parse and Firebase, which are two solutions that meet the

company's selection criteria. This work suggests that the company use Firebase, which appears to be the solution that meets all the recommendations and the most accessible for the company. Because of the functionalities presented through this work, Firebase is a complete BaaS solution that offers several other attractive functionalities for Red Nuclear Monkey, such as the distribution of the application for customers or testers. The company seems happy with the result of the work proposed by this thesis and looks forward to releasing the final version of Alacrity on the App Store and Google Play Store using the features provided by Firebase.

7.1 Limitations and recommendations for future research

This thesis has limitations regarding the literature search to illustrate the motivations of companies in their choice to use a Backend-as-a-Service solution or a custom backend. Indeed, few academic sources address this subject, and the motivations presented by this thesis are mainly derived from blog articles written by platforms offering BaaS solutions. Thus, the judgment of these platforms can be questioned.

Also, as the thesis presented, the Flutter framework is expanding rapidly and does not yet have all the necessary tools to compare multiple Backend-as-a-Service solutions. At the time of development of the Alacrity application, two BaaS solutions met company criteria and were easily usable for Flutter. However, as explained in Chapter 4, AWS Amplify is now available for Flutter and comes with extensive documentation. With the constant evolution of the Flutter framework, it would intrigue to see the new BaaS platforms available for Flutter. There is no doubt that Flutter is becoming a popular framework and that other BaaS solutions, like AWS Amplify, will follow Firebase and Parse's lead. Thus, other studies can perform a new comparison of BaaS solutions and provide the best of these solutions to develop a complete application efficiently with Flutter.

Bibliography

[1] Abir, H., and Khaled, R. A smart city system using backend as a service approach: Biskra city case study. In *2018 Fifth International Symposium on Innovation in Information and Communication Technology (ISIICT)* (2018), pp. 1–7. https://ieeexplore-ieee-org.libproxy.aalto.fi/stamp/stamp.jsp?tp=&arnumber=8613725 (in English). Accessed 26.08.2020.

[2] Al-Fedaghi, S., and Alrashed, A. Schematizing uml use cases. pp. 35–41. https://www.researchgate.net/publication/269328789_Schematizing_UML_use_cases (in English). Accessed 12.08.2020.

[3] Alessio Spallino. Native versus hybrid mobile application development for professional membership services. Master's thesis, Aalto University, School of Science, 2018. https://aaltodoc.aalto.fi/bitstream/handle/123456789/33767/master_Spallino_Alessio_2018.pdf?sequence=1&isAllowed=y (in English). Accessed 12.08.2020.

[4] Artyom Dogtiev. App stores list (2019), 2019. https://www.businessofapps.com/guide/app-stores-list/ (in English). Accessed 6.07.2020.

[5] Awel Eshetu Fentaw. Cross platform mobile application development: a comparison study of react native vs flutter. Master's thesis, University of Jyväskylä, 2020. https://jyx.jyu.fi/bitstream/handle/123456789/70969/URN%3aNBN%3afi%3ajyu-202006295155.pdf?sequence=1&isAllowed=y (in English). Accessed 19.08.2020.

[6] Bartosz, Aleksandra. Backend as a service (baas) vs. custom backend: Which one to choose and why?, 2019. https://www.merixstudio.com/blog/backend-service-baas-vs-custom-backend/ (in English). Accessed 29.06.2020.

[7] Churi, P. P., Wagh, S., Kalelkar, D., and Kalelkar, M. Model-view-controller pattern in bi dashboards: Designing best practices. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)* (2016), pp. 2082–2086. https://ieeexplore-ieee-org.libproxy.aalto.fi/stamp/stamp.jsp?tp=&arnumber=7724633 (in English). Accessed 04.08.2016.

[8] Cuong Le. Design patterns implementation in video game programming, 2016. https://www.theseus.fi/bitstream/handle/10024/110510/Cuong_Le.pdf?sequence=1&isAllowed=y (in English). Accessed 05.08.2020.

[9] Ding, J., Song, W., and Zhang, D. An approach for modeling and analyzing mobile push notification services. In *2014 IEEE International Conference on Services Computing* (2014), pp. 725–732. https://ieeexplore-ieee-org.libproxy.aalto.fi/stamp/stamp.jsp?tp=&arnumber=6930601 (in English). Accessed 16.09.2020.

[10] Dobing, B., and Parsons, J. How uml is used. *Commun. ACM 49* (05 2006), 109–113. https://www.researchgate.net/publication/220427796_How_UML_is_used (in English). Accessed 12.08.2020.

[11] Falih, N., and Firdaus, A. Measuring performance, functionality and portability for mobile hybrid application. In *2019 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS)* (2019), pp. 195–200.

[12] Gamma, E., Helm, R., Johnson, R. Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education.

[13] George Batschinski. Backend as a service â what is a baas? https://blog.back4app.com/backend-as-a-service-baas/ (in English). Accessed 01.09.2020.

[14] Ian Blair. Mobile app download and usage statistics (2020), 2020. https://buildfire.com/app-statistics/ (in English). Accessed 29.06.2020.

[15] inVerita. Flutter vs native vs react-native: Examining performance, 2020. https://medium.com/swlh/flutter-vs-native-vs-react-native-examining-performance-31338f081980 (in English). Accessed 21.08.2020.

[16] inVerita. Flutter vs react native vs native: Deep performance comparison, 2020. https://medium.com/swlh/flutter-vs-react-native-vs-native-deep-performance-comparison-990b90c11433 (in English). Accessed 21.08.2020.

[17] Ishihara, T., Hotta, K., Higo, Y., and Kusumoto, S. Reusing reused code. In *2013 20th Working Conference on Reverse Engineering (WCRE)* (2013), pp. 457–461. https://ieeexplore-ieee-org.libproxy.aalto.fi/stamp/stamp.jsp?tp=&arnumber=6671322 (in English). Accessed 05.08.2020.

[18] Iulia-Maria, T., and Ciocarlie, H. Best practices in iphone programming: Model-view-controller architecture â carousel component development. In *2011 IEEE EUROCON - International Conference on Computer as a Tool* (2011), pp. 1–4. https://ieeexplore-ieee-org.libproxy.aalto.fi/stamp/stamp.jsp?tp=&arnumber=5929308 (in English). Accessed 04.08.2020.

[19] J.Clement. Device usage of facebook users worldwide as of april 2020, 2020. https://www.statista.com/statistics/377808/distribution-of-facebook-users-by-device/ (in English). Accessed 29.06.2020.

[20] J.Clement. Number of monthly active facebook users worldwide as of 1st quarter 2020, 2020. https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/ (in English). Accessed 29.06.2020.

[21] Kin Lane. Overview of the backend as a service (baas) space, 2015. https://apievangelist.com/2013/05/03/overview-of-the-backend-as-a-service-baas-space/ (in English).

[22] Krueger, C. W. Software reuse. *ACM Comput. Surv. 24*, 2 (June 1992), 131â183. http://lab.dnict.vn/Resource/Courses/UnZip/16/20121214_21/data/downloads/4.pdf (in English). Accessed 05.08.2020.

[23] Leff, A., and Rayfield, J. T. Web-application development using the model/view/controller design pattern. In *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference* (2001), pp. 118–127. https://ieeexplore-ieee-org.libproxy.aalto.fi/stamp/stamp.jsp?tp=&arnumber=950428 (in English). Accessed 04.08.2020.

[24] Mai Hoang Bich Tram. (firebase, 2019. https://www.theseus.fi/bitstream/handle/10024/263641/Mai_Tram.pdf?sequence=2&isAllowed=y (in English). Accessed 04.09.2020.

[25] Matilda Olsson. A comparison of performance and looks between flutter and native applications. Master's thesis, Blekinge Institute of Technology, 2020. http://www.diva-portal.org/smash/get/diva2:1442804/FULLTEXT01.pdf (in English). Accessed 19.08.2020.

[26] Michael Gonsalves. Evaluating the mobile development frameworks apache cordova and flutter and their impact on the development process and application characteristics. Master's thesis, Faculty of California State University, 2018. http://csuchico-dspace.calstate.edu/bitstream/handle/10211.3/211157/Gonsalves_Thesis_Fall18.pdf?sequence=1 (in English). Accessed 21.08.2020.

[27] Mohammad Abdullah Atik. Applying software design pattern on ios application, 2014. https://www.theseus.fi/bitstream/handle/10024/83088/Final_Thesis_revised.pdf?sequence=1&isAllowed=y (in English). Accessed 04.08.2020.

[28] Ola Dahl. Exploring end user's perception of flutter mobile apps. Master's thesis, Malmö University, 2019. http://ls00012.mah.se/bitstream/handle/2043/30443/Ola%20Dahl.pdf?sequence=1&isAllowed=y (in English). Accessed 19.08.2020.

[29] Pan, Z., Liang, X., Zhou, Y. C., Ge, Y., and Zhao, G. T. Intelligent push notification for converged mobile computing and internet of things. In *2015 IEEE International Conference on Web Services* (2015), pp. 655–662. https://ieeexplore-ieee-org.libproxy.aalto.fi/stamp/stamp.jsp?tp=&arnumber=7195627 (in English). Accessed 16.09.2020.

[30] Phongtraychack, A., and Dolgaya, D. Evolution of mobile applications. *MATEC Web of Conferences 155* (01 2018), 01027. https://www.researchgate.net/publication/323447470_Evolution_of_Mobile_Applications.

[31] Phu Nguyen. Mobile backend as a service: The pros and cons of parse, 2016. https://www.theseus.fi/bitstream/handle/10024/117483/Nguyen_Phu.pdf?sequence=2 (in English). Accessed 01.09.2020.

[32] Professor John.F Clark. History of mobiles apps, 2012. https://www.uky.edu/~jclark/mas490apps/History%20of%20Mobile%20Apps.pdf (in English). Accessed 29.06.2020.

[33] RedHat. Red hat survey reveals enterprise mobility hiring priorities, 2015. https://www.redhat.com/en/about/press-releases/red-hat-survey-reveals-enterprise-mobility-hiring-priorities (in English). Accessed 29.06.2020.

[34] S. O'Dea. Number of smartphone users worldwide from 2016 to 2021, 2020. https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/ (in English). Accessed 29.06.2020.

[35] ScaleTech. Why to hybrid app development instead of native app development, 2020. https://medium.com/@ScaleTech/the-reason-behind-the-rise-in-hybrid-app-development-f089ae80222c (in English). Accessed 29.06.2020.

[36] Siegfried Rasthofer, Steven Arzt, Robert Hahn Max Kolhagen. (in)security of backend-as-a-service. In *Black Hat Europe 2015* (2015). https://www.blackhat.com/docs/eu-15/materials/eu-15-Rasthofer-In-Security-Of-Backend-As-A-Service-wp.pdf (in English). Accessed 01.09.2020.

[37] Sofiya Merenych. The hard choice between mobile backend as a service and custom backend explained with pizza, 2019. https://clockwise.software/blog/choice-between-mobile-backend-as-a-service-and-custom-backend/ (in English). Accessed 01.09.2020.

[38] Sophia Martin. Flutter vs react native- which is the best choice for 2020?, 2020. https://codeburst.io/flutter-or-react-native-which-is-the-best-choice-for-2020-355d9473edde (in English). Accessed 20.08.2020.

[39] Thanh Tran. Flutter native performance and expressive ui/ux, 2020. https://www.theseus.fi/bitstream/handle/10024/336980/Thanh_Tran.pdf?sequence=2&isAllowed=y (in English). Accessed 19.08.2020.

[40] Ville Kähärä. Implementation of interactive application for maretarium aquarium, 2019. https://www.theseus.fi/bitstream/handle/

10024/169493/Kahara_Ville.pdf?sequence=2&isAllowed=y (in English). Accessed 10.09.2020.

[41] Wang, X., Xu, B., and Gu, R. The application of code reuse technology based on the mvc framework. In *2013 International Conference on Computer Sciences and Applications* (2013), pp. 534–537. https://ieeexplore-ieee-org.libproxy.aalto.fi/stamp/stamp.jsp?tp=&arnumber=6835657&tag=1 (in English). Accessed 05.08.2020.

[42] Wenhao Wu. React native vs flutter, cross-platform mobile application frameworks, 2018. https://www.theseus.fi/bitstream/handle/10024/146232/thesis.pdf?sequence=1&isAllowed=y (in English). Accessed 20.08.2020.